Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Д. М. Романенко, С. А. Осоко

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PHP

Рекомендовано
учебно-методическим объединением
по химико-технологическому образованию
в качестве учебно-методического пособия
для студентов учреждений высшего образования
по специальности 1-47 01 02 «Дизайн электронных и веб-изданий»

Минск 2021

УДК 655.2/.3(075.8)
ББК 37.8я73
Р17

Рецензенты:

кафедра программного обеспечения информационных технологий Белорусского государственного университета информатики и радиоэлектроники (заведующий кафедрой кандидат технических наук, доцент *Н. В. Лапицкая*);
заведующий кафедрой информационных технологий и моделирования экономических процессов Белорусского государственного аграрного технического университета кандидат педагогических наук, доцент *О. Л. Сапун*

Романенко, Д. М.

Р17 Программирование на языке PHP : учеб.-метод. пособие для студентов специальности 1-47 01 02 «Дизайн электронных и веб-изданий» / Д. М. Романенко, С. А. Осоко. – Минск : БГТУ, 2021. – 256 с.

ISBN 978-985-530-855-4.

Учебно-методическое пособие предназначено для самостоятельной подготовки студентов специальности «Дизайн электронных и веб-изданий» по дисциплине «Веб-программирование электронных изданий» и включает теоретические сведения об основах программирования на языке PHP. В книге приведены примеры решения типовых задач в соответствии с программой курса.

Изложенный в издании материал позволяет расширить общие знания о веб-программировании, разработке веб-ресурсов и может быть интересен студентам других специальностей.

УДК 655.2/.3(075.8)
ББК 37.8я73

ISBN 978-985-530-855-4

ПРЕДИСЛОВИЕ

Современный мир уже невозможно представить без интернета. Он повсюду в жизни каждого человека: в интернете ищется вся необходимая информация, в нем с применением облачных технологий осуществляется хранение данных; различные мессенджеры, социальные сети, а также электронная почта используются как средства общения и обмена информацией, разнообразые интернет-сервисы позволяют приобретать товар, проводить финансовые операции и т. д.

Сегодня даже СМИ постепенно переходят в пространство интернета, публикуя в нем свои статьи, новостные материалы, интервью и т. д. Жизнь в таком мире требует существования качественных web-сайтов и сервисов.

Статические web-сайты скучны с точки зрения преподнесения информации и неудобны в управлении. Намного интереснее динамические web-сайты, поскольку их содержимое изменяется. Большая статическая html-страница, на которой отображаются материалы, например все статьи, как ранее было в печатных изданиях, изображения, описания и другая информация, неудобна в употреблении и бесконечно долго загружается. А динамическая web-страница, например со статьями, их кратким содержанием, на которой можно искать и отбирать статьи по разным категориям или ключевым словам, оказывается более удобной для читателей, оперативно преподносящей информацию, что в итоге приводит к расширению пользователей сайта.

Язык программирования PHP упрощает создание динамических web-сайтов. Он позволяет решать самые разные задачи создания интерактивного содержимого: составление каталога новостных статей, фотоальбома, календаря событий и даже организация блога.

Изучив данное пособие, вы будете способны справиться с задачей построения динамического web-сайта.

ГЛАВА 1

ЯЗЫК PHP – ЯЗЫК СЦЕНАРИЕВ

PHP – это широко используемый язык сценариев общего назначения с открытым исходным кодом, т. е. PHP – это язык программирования, специально разработанный для написания web-приложений (сценариев), исполняющихся на web-сервере. Синтаксис языка берет начало из C, Java и Perl.

Преимуществом PHP является предоставление web-разработчикам возможности быстрого создания динамически генерируемых web-страниц. Среди преимуществ языка PHP перед такими языками, как Perl и C можно выделить возможность создания HTML-документов с внедренными командами PHP, поддержку широкого круга баз данных.

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

PHP доступен для большинства операционных систем, включая Linux, многие модификации Unix (HP-UX, Solaris и OpenBSD), Microsoft Windows, Mac OS X, RISC OS и др. В PHP включена поддержка большинства современных web-серверов, таких как Apache, Microsoft Internet Information Server, серверов Netscape и т. д.

1.1. Основы языка PHP. Сценарии

В целом программы PHP могут выполняться двумя способами: как сценарное приложение web-сервером и как консольные программы. Поскольку нашей задачей является программирование web-приложений и их интерфейсов, в дальнейшем будем говорить только про первый способ.

Рассмотрим процесс выполнения PHP-сценария при обращении браузера к серверу. Вначале браузер запрашивает страницу с расширением .php, после чего web-сервер пропускает программу через машину PHP и выдает результат в виде html-кода. Причем если взять стандартную страницу html, изменить расширение на .php и пропустить ее через машину PHP, последняя просто перешлет ее пользователю без изменений. Для включения в этот файл команды PHP необходимо заключить команды PHP в специальные теги. Различают 4 вида тегов (они эквивалентны и можно использовать любые):

1. Инструкция обработки XML

```
<?PHP
...
?>
```

2. Инструкция обработки SGML

```
<?
...
?>
```

3. Инструкция обработки сценариев HTML

```
<script language = "PHP">
...
</script>
```

4. Инструкция в стиле ASP

```
<%
...
%>
```

Необходимо отметить, что в PHP7 убраны или более некорректны третий и четвертый варианты открывающихся / закрывающихся тегов.

1.2. Использование web-сервера для выполнения PHP-сценариев

Для использования языка PHP необходимо сначала установить один из web-серверов, например Wamp (http://www.wampserver.com/ru/), OpenServer (https://ospanel.io), Denwer (http://www.denwer.ru/), XAMPP (https://www.apachefriends.org/ru/index.html) и т. д. Все они имеют выделенный каталог для размещения проектов, написанных

на языке PHP. Так, например, для web-сервера Wamp по умолчанию это C:\wamp\www, а для OpenServer – C:\OpenServer\OSPanel\domains. В данном каталоге необходимо создать каталог проекта, например test.dev или просто test (отметим, что некоторые web-серверы не допускают использования многоуровневого вложения доменных имен). По умолчанию стартовым файлом будет index.php. Выведем в данном файле информацию об используемой версии PHP с помощью команды phpinfo() (рис. 1.1).

```
<?php
    phpinfo();
?>
```

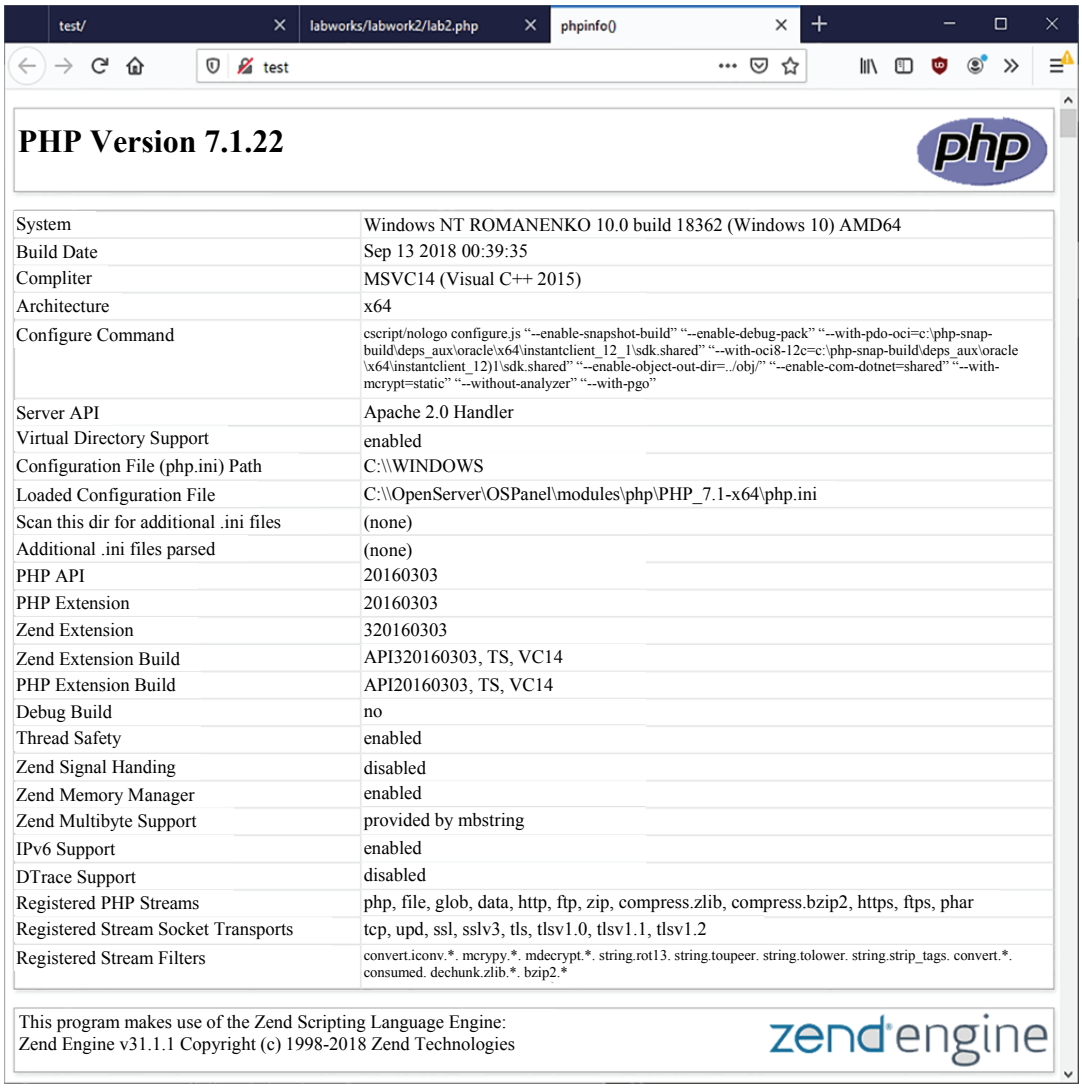

Рис. 1.1. Результаты выполнения файла index.php

В версии PHP7 появилась возможность выполнять скрипты с помощью встроенного web-сервера. Для того чтобы воспользоваться этой возможностью, необходимо скачать установочный файл PHP 7.4.7 с сайта www.php.net для вашей операционной системы. Из предложенных вариантов следует выбрать дистрибутив x86 для 32-битной системы или x64 для 64-битной системы. Вариант Thread Safe служит для безопасного выполнения PHP в рамках одного системного процесса, обычно при использовании внешнего сервера, такого как Apache. Если использовать встроенный сервер, лучше воспользоваться вариантом Non Thread Safe. Скачиваем zip-архив с выбранным вариантом релиза.

Следует обратить внимание на аббревиатуру VC15, обозначающую версию Visual Studio (2015), при помощи которой был скомпилирован дистрибутив. Для того чтобы запустить проект, необходимо загрузить ***английский вариант*** распространяемого пакета Visual C++ для Visual Studio, который содержит нужные динамические библиотеки (https://www.microsoft.com/en-us/download/details.aspx?id=48145).

После распаковки zip-архива, например в папку C:\PHP, проверим версию PHP. Для этого в командной строке набираем комманды

```
    >cd C:\PHP
    >PHP –v
    PHP 7.4.3 (cli) (built: Feb 18 2020 17:29:57) ( NTS
Visual C++ 2017 x64 )
    Copyright (c) The PHP Group
    Zend Engine v3.4.0, Copyright (c) Zend Technologies
    >
```

Проверим работоспособность встроенного сервера. Для этого создадим текстовый файл с именем index.php в папке D:\LabWorks со следующим содержанием:

```
    <?PHP
      PHPinfo();
    ?>
```

После этого в командной строке вводим команду для перехода в папку с файлом index.php.

```
    cd /D D:\LabWorks
```

Запускаем встроенный web-сервер на 4000-м порту, записав в командной строке команду

```
C:\PHP\PHP -S localhost:4000
```

Теперь в адресной строке браузера набираем команду

```
localhost:4000/index.php
```

Результат выполнения команды представлен на рис. 1.2.

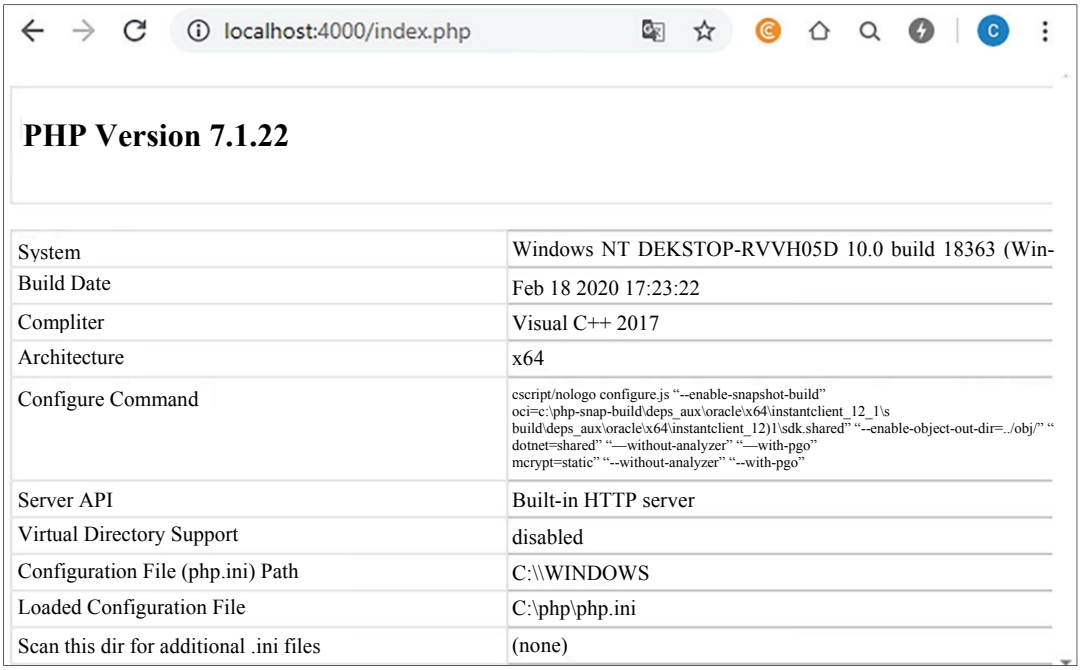

Рис. 1.2. Результаты выполнения файла index.php с помощью встроенного web-сервера

Пролистав страницу вниз, можно получить полную информацию об установленной версии PHP, например о каталоге, в котором установлен дистрибутив, установленных библиотеках, их версиях и др.

1.3. Первый PHP-скрипт

Итак, возьмем в качестве основного сервера OpenServer. В C:\OpenServer\OSPanel\domains создадим каталог test и в нем

файл index.php со следующим содержимым (оператор echo позволяет выводить на экран).

```
<!DOCTYPE html>
<html>
<head>
   <title>Пример</title>
</head>
<body>
   <?php
   echo 'Привет! Я первый скрипт, написанный на языке
PHP.';
   ?>
</body>
</html>
```

Обратите внимание, что html-код корректно обрабатывается интерпретатором PHP. Главная особенность языка PHP (кстати, чрезвычайно удобная): PHP-скрипт может вообще не отличаться от обычного html-документа. Сам код сценария начинается после открывающего тега **<?PHP** и заканчивается закрывающим **?>**. Между этими двумя тегами текст интерпретируется как программа и в html-документ не попадает. Если же программе нужно что-то вывести, она должна воспользоваться оператором *echo*.

PHP устроен так, что любой текст, который расположен вне программных блоков, ограниченных **<?PHP** и **?>**, выводится в браузер непосредственно. В этом и заключается главная особенность PHP.

Далее откроем любой web-браузер и перейдем на страницу с адресом http://test/, где test – это название проекта (корневого каталога проекта). Отметим, что некоторые web-серверы требуют другой вариант URL-строки, а именно http://localhost/test. В итоге получим (рис. 1.3):

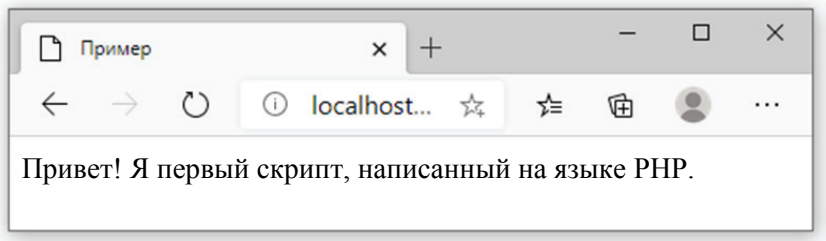

Рис. 1.3. Первый скрипт на PHP

Для дальнейшей работы данную страницу можно не закрывать, а при необходимости обновлять. Для разработки приложений на языке PHP удобно пользоваться какой-либо популярной IDE-программой, например PHPStorm, Sublime Text, Visual Studio Code, Eclipse PDT, Zend Studio и т. д. Пример применения Sublime Text представлен на рис. 1.4.

При необходимости просмотра выполнения какого-либо PHP-файла (не index.php) (рис. 1.5) следует в URL-строке указать путь к нему относительно каталога проекта (рис. 1.6): http://test/lab1/lab1.php.

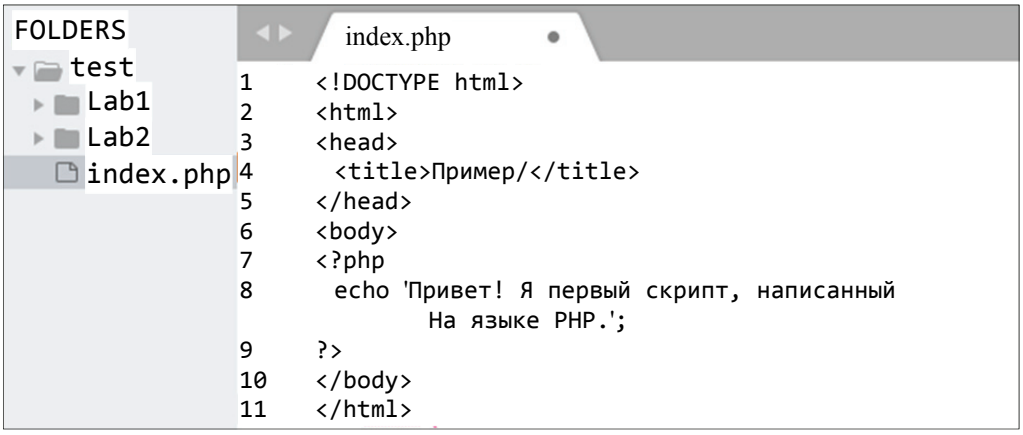

Рис. 1.4. Использование Sublime Text для разработки скриптов на PHP

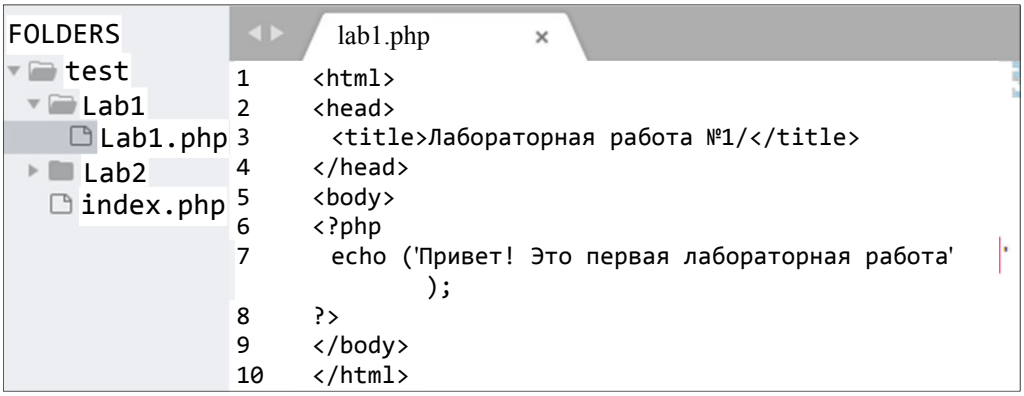

Рис. 1.5. Пример файла lab1.php

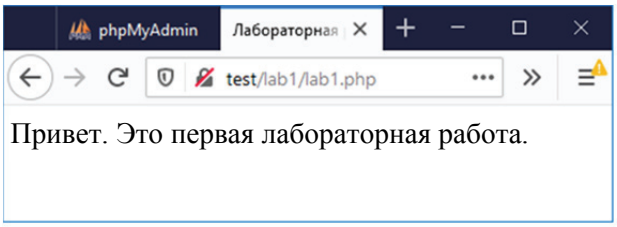

Рис. 1.6. Пример выполнения файла lab1.php

Инструкции разделяются также, как и в C или Perl: каждое выражение заканчивается точкой с запятой. Закрывающий тег (?>) подразумевает конец инструкции, поэтому два следующих фрагмента кода эквиваленты (многострочный и однострочный варианты кода).

```
<?PHP
 echo "Это тест";
?>
```

```
<?PHP echo "Это тест" ?>
```

Код необходимо сохранить в файле с расширением .php.

1.4. Комментарии в PHP-скриптах

Написание практически любого скрипта не обходится без комментариев. PHP поддерживает комментарии в стиле C, C++ и оболочки Unix.

```
<?PHP
echo "Это тест"; // Это однострочный комментарий в стиле c++

/* Это многострочный комментарий
    echo "Это еще один тест";
   еще одна строка комментария */

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

Однострочные комментарии идут только до конца строки, (признаком которой является символ «;») или текущего блока PHP-кода в зависимости от того, что идет перед ними.

```
<h1>Это <?PHP # echo "простой";?> пример</h1>
```

Заголовок вверху выведет «Это пример». Но будьте внимательны, следите за отсутствием вложенных 'C'-комментариев, они могут появиться во время комментирования больших блоков.

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

Это означает, что HTML-код после // ?> БУДЕТ напечатан: ?> выводит из режима PHP и возвращает в режим HTML, но // не позволяет этого сделать.

Практическая часть

Рассмотрим пример совместного использования тегов HTML и языка программирования PHP. Пусть необходимо вывести «Hello!!!» по центру.

```
    <p align="center">
        <?PHP
            echo "Hello!!!";
        ?>
    </p>
```

Результаты выполнения скрипта представлены на рис. 1.7.

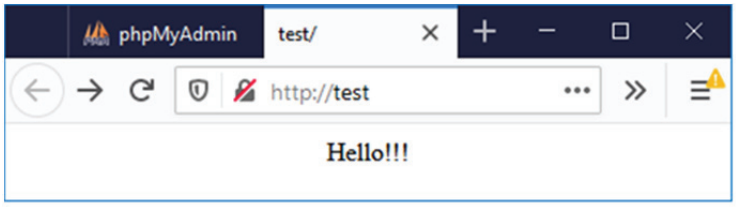

Рис. 1.7. Пример использования PHP и HTML

Тот же результат будет получен и при выполнении следующего кода:

```
<?PHP
    echo "<p align='center'>";
    echo "Hello!!!";
    echo "</p>";
?>
```

Обратите внимание, что теги html, задающие разметку, можно использовать и внутри PHP-скрипта, но их в таком случае надо выводить на экран, например, с помощью команды echo.

PHP-скрипты можно писать многократно.

```
<p align="center"> <?PHP echo "Hello!!!" ?> <p>
<hr><br>
<p     align="center">     <?PHP     echo     "Hello!!!"
?>      <?PHP  echo  "Hel-
lo!!!" ?> <p>
<hr><br>
<p align="center"> <?PHP echo "Hello!!!" ?> <p>
```

Результаты выполнения скрипта представлены на рис. 1.8.

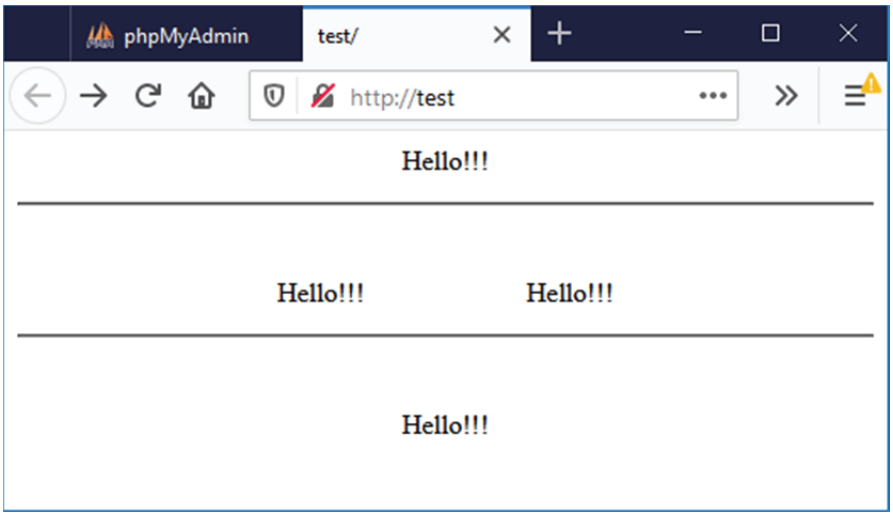

Рис. 1.8. Пример использования тегов внутри скрипта

Но лучше не стараться объединять множество строк в одну, а «красиво», с использованием табуляции писать в несколько строк. Такой код легче читается.

```
<p align="center">
   <?PHP
        echo "Hello!!!";
    ?>
<p>
<hr><br>
<p align="center">
    <?PHP
        echo "Hello!!!";
```

```
    ?>
          
    <?PHP
        echo "Hello!!!";
    ?>
<p>
<hr><br>
<p align="center">
    <?PHP
        echo "Hello!!!";
    ?>
<p>
```

Также приведем еще один пример использования различных html-тегов совместно с PHP.

```
<p align="center">
    <span style="color: red; font-size: 2em">
        <?PHP
            echo "Hello!!!";
        ?>
    </span>
<p>
<hr>
<br>
<p align="center">
    <span style="color: green; font-size: 3em">
        <?PHP
            echo "Hello!!!";
        ?>
              
        <?PHP
            echo "Hello!!!";
        ?>
    </span>
<p>
<hr>
<br>
<p align="center">
    <span style="color: blue; font-size: 2em">
        <?PHP
            echo "Hello!!!";
        ?>
    </span>
<p>
```

Результаты выполнения скрипта представлены на рис. 1.9.

Рис. 1.9. Пример использования html-тегов внутри скрипта

Можно также использовать CSS совместно с PHP для придания web-странице требуемого вида, например, применять разные шрифты, цвета и т. д.

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>Стили</title>

    <style type="text/css">
        H1 {
            font-size: 120%; /* Размер шрифта */
            font-family: Verdana, Arial, Helvetica,
sans-serif; /* Семейство шрифта */
            color: #FF0000; /* Цвет текста */
            text-align: left;
        }
    </style>
    <style type="text/css">
        H2 {
            font-size: 160%; /* Размер шрифта */
            font-family: Verdana, Arial, Helvetica,
sans-serif; /* Семейство шрифта */
            color: #00FF00; /* Цвет текста */
            text-align: center;
        }
    </style>
```

```
    <style type="text/css">
        H3 {
            font-size: 200%; /* Размер шрифта */
            font-family: Verdana, Arial, Helvetica,
sans-serif; /* Семейство шрифта */
            color: #0000FF; /* Цвет текста */
            text-align: right;
        }
    </style>
</head>
<body>
    <?PHP
        echo "<H1>";
        echo "HELLO WORLD!";
        echo "</H1>";

        echo "<H2>";
        echo "HELLO WORLD!";
        echo "</H2>";

        echo "<H3>";
        echo "HELLO WORLD!";
        echo "</H3>";

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

Результаты выполнения скрипта представлены на рис. 1.10.

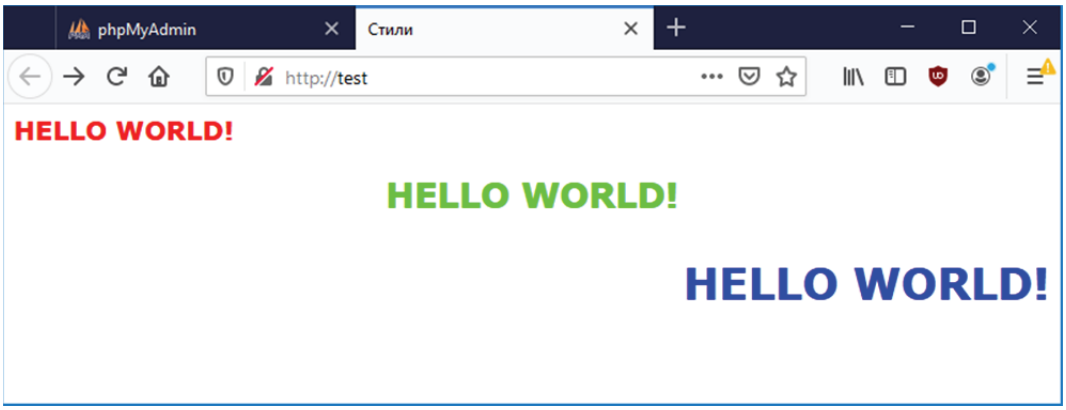

Рис. 1.10. Пример использования html-тегов и CSS вместе с PHP

Таким образом можно совместно применять html-теги и CSS вместе с PHP.

Лабораторная работа № 1

В рамках данной дисциплины предполагается выполнение всех лабораторных работ в виде одного проекта. Каждая лабораторная будет сделана в виде отдельного PHP файла. Желательно, чтобы данный файл размещался в отдельном каталоге. Если же какая-либо лабораторная работа будет требовать использования нескольких файлов, то они должны размещаться в отдельном каталоге. Таким образом, структуру данного проекта можно представить следующим образом (рис. 1.11).

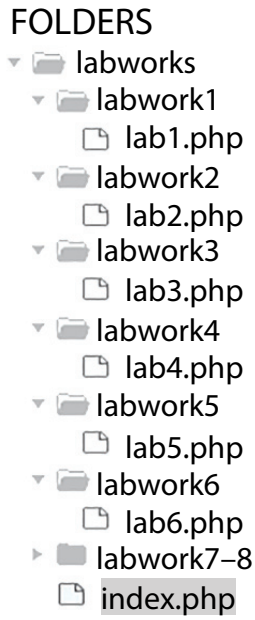

Рис. 1.11. Пример структуры папок для проекта labworks

Код стартового файла зависит от количества предполагаемых лабораторных работ, а также используемых стилей CSS. Так, если бы в проекте было всего три лабораторные работы, то код стартового файла labworks/index.php мог бы быть следующим:

```
<html>
 <head>
  <meta charset="utf-8">
  <title>Лабораторные работы</title>

  <style>
      .title{
        text-align: center;
        border-bottom: solid;
        padding-bottom: 20px;
      }
      .menu{
        margin-top: 50px;
      }
       .c {
        border: 1px solid #333; /* Рамка */
        display: inline-block;
        padding: 10px 20px; /* Поля */
```

```
        margin: 10px 20px;
        text-decoration: none; # Убираем подчеркивание
        color: #000; /* Цвет текста */
      }
       .c:hover {
        box-shadow: 0 0 5px rgba(0,0,0,0.3); /* Тень */
        background: linear-gradient(to bottom, #fcfff4,
#e9e9ce); /* Градиент */
        color: #a00;
      }
  </style>
 </head>

<body>
    <div class="title">
        <h1>
            ЛАБОРАТОРНЫЕ РАБОТЫ
        </h1>
        <h3>
            Выполнил: Иванов Иван Иванович
        </h3>
    </div>
    <div class="menu">
       <a href="labwork1/lab1.php"
class="c">Лабораторная работа № 1</a>
        <a href="labwork2/lab2.php"
class="c">Лабораторная работа № 2</a>
        <a href="labwork3/lab3.php"
class="c">Лабораторная работа № 3</a>
        <a href="labwork4/lab4.php"
class="c">Лабораторная работа № 4</a>
        <a href="labwork5/lab5.php"
class="c">Лабораторная работа № 5</a>
        <a href="labwork6/lab6.php"
class="c">Лабораторная работа № 6</a>
        <a href="labwork7-8/lab7-8.php"
class="c">Лабораторная работа №7-8</a>
    </div>

 </body>
</html>
```

На рис. 1.12 представлен результат выполнения данного кода.

В соответствии со структурой, представленной на рис. 1.10, все файлы лабораторной работы хранятся в отделной папке.

Стартовым файлом для каждой работы также является файл lab1.php, lab2.php и т. д., расположенный в соответствующей папке (названия файлов могут быть и другие, при этом необходимо контролировать их в ссылках на главной странице проекта). Структура данного файла будет зависеть от заданий, выданных студенту преподавателем. Условно, если студенту выдана для выполнения только одна задача, пусть файл лабораторной работы № 1 выглядит следующим образом (рис. 1.13).

На данный файл в проекте переходим щелчком по кнопке «Лабораторная работа № 1» (рис. 1.14).

Если в рамках лабораторной работы требуется выполнить, например, несколько заданий, то соответствующая стартовая страница лабораторной работы может по аналогии содержать несколько кнопок, ведущих к просмотру результатов выполнения отдельной задачи. На рис. 1.15 представлен код реализации ссылок на станицы с решением задач (разметка, стили и т. д. не представлены), а на рис 1.16 – результат выполнения данного кода.

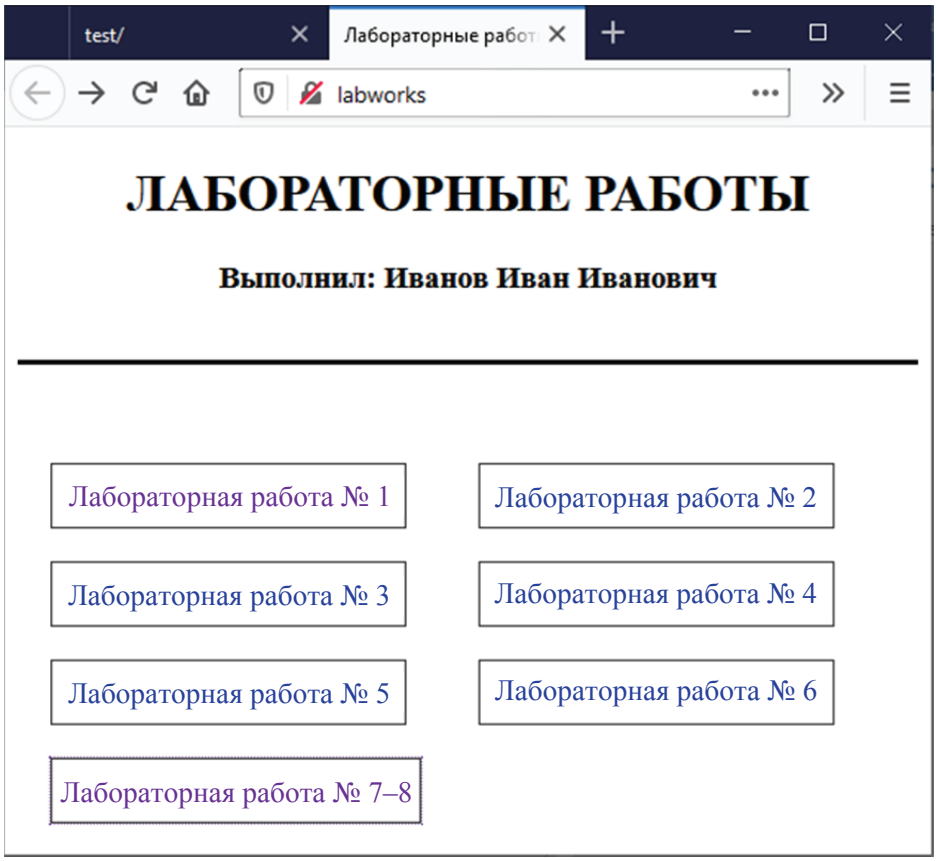

Рис. 1.12. Пример стартовой страницы проекта labworks

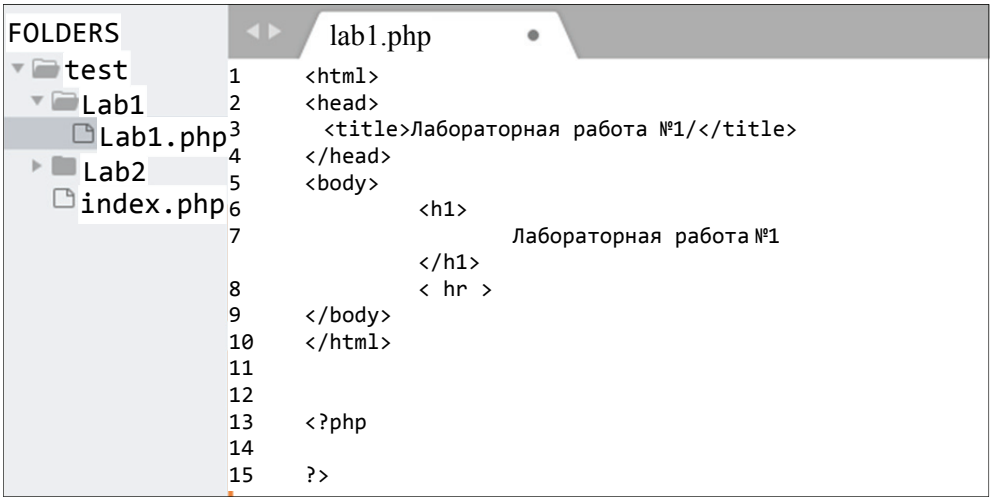

Рис. 1.13. Пример кода файла лабораторной работы

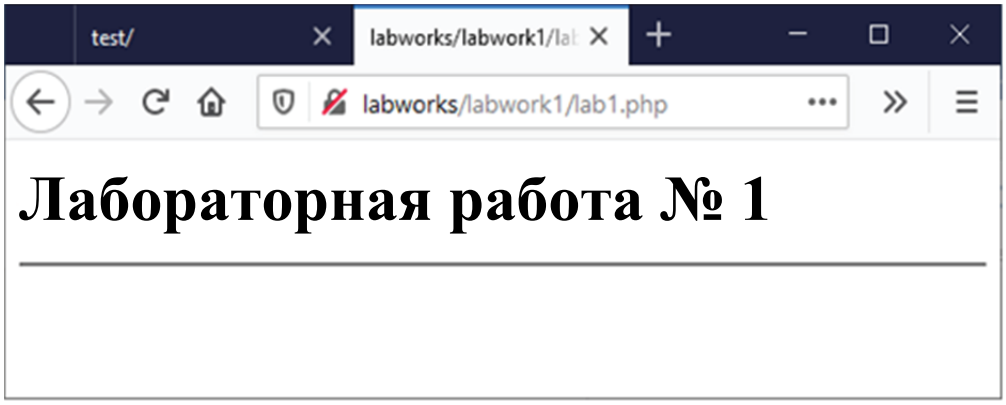

Рис. 1.14. Пример условной лабораторной работы № 1

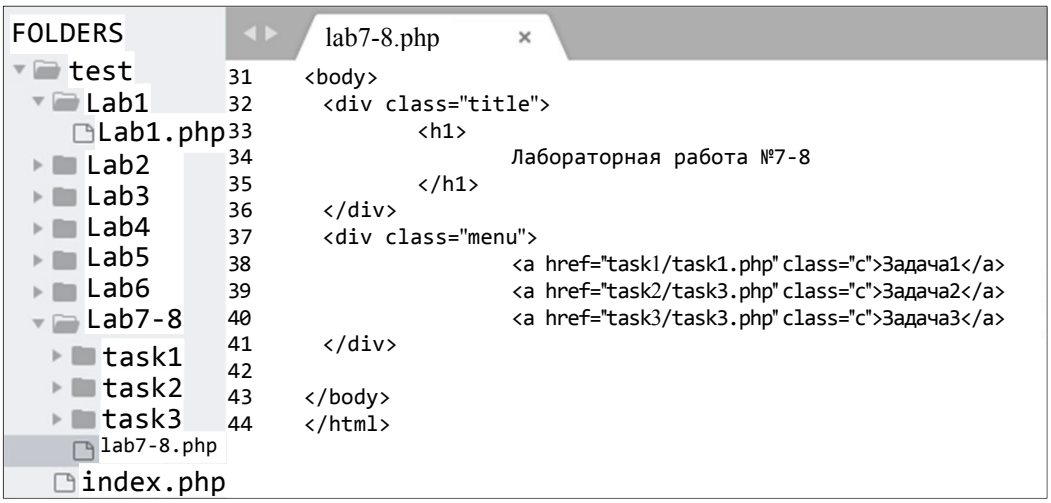

Рис. 1.15. Пример кода условной лабораторной работы № 7–8 со сылками на три задачи

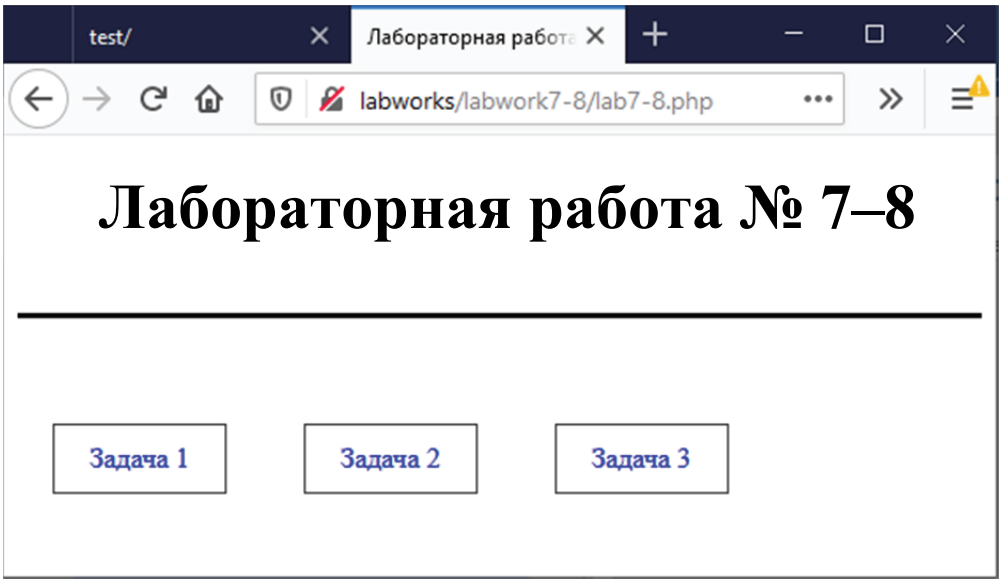

Рис. 1.16. Демотрация вополнения кода условной лабораторной работы № 7–8 со ссылками на три задачи

Задание. Таким образом, в рамках данной лабораторной работы необходимо разработать базовую структуру проекта лабораторных работ labworks, оформить с использованием html, CSS и, если надо, JavaScript стартовую страницу проекта, а также стартовые страницы соответствующих лабораторных работ.

ГЛАВА 2

ПЕРЕМЕННЫЕ В PHP. ТИПЫ ДАННЫХ

2.1. Общие понятия о переменных в PHP

Как и в любом другом языке программирования, в PHP существует такое понятие, как переменная. При программировании на PHP можно не скупиться на объявление новых переменных. Принципы экономии памяти, которые были актуальны несколько лет назад, сегодня в расчет не принимаются. Однако при хранении в переменных больших объемов памяти лучше удалять неиспользуемые переменные с помощью оператора *unset ()*.

Вообще, переменная – это область оперативной памяти, доступ к которой осуществляется по имени. Все данные, с которыми работает программа, хранятся в виде переменных (исключение – константа, которая, впрочем, может содержать только число или строку). Такого понятия, как указатель (как в C), в PHP не существует. При присвоении переменная копируется один-в-один, какую бы сложную структуру она не имела. Тем не менее в PHP существует понятие ссылок – жестких и символических.

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

В официальной документации PHP указано, что имя переменной может состоять не только из букв латиницы и цифр, но также и из любых символов, код ASCII которых старше 127, – в частности, и из символов кириллицы, т. е. русских букв! Однако не рекомендуется применять кириллицу в именах переменных хотя бы из-за того, что в различных кодировках ее буквы имеют различные коды. Впрочем, можно поэкспериментировать и делать так, как вам будет удобно.

Вывод: переменные в PHP – это особые объекты, которые могут содержать в буквальном смысле все, что угодно.

Приведем некоторые примеры переменных в PHP, а также пример удаления переменных.

```
<?PHP
$var = "Bob";
$Var = "Joe";
echo "переменная var=$var, а переменная Var=$Var";
// выведет "Bob, Joe"
?>
```

Результаты выполнения скрипта представлены на рис. 2.1.

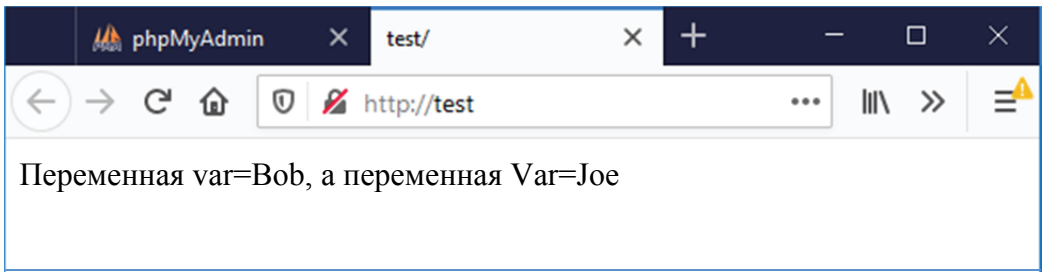

Рис. 2.1. Пример использования переменных

Приведем еще несколько примеров работы с переменными (создание, удаление) на языке PHP.

```
<?PHP
$4site = 'not yet';     // неверно; начинается с цифры
$_4site = 'not yet';    // верно; начинается с символа
подчеркивания
$тдуte = 'mansikka';    // верно; 'д' это (Дополнитель-
ный) ASCII 228.
?>
```

```
<?PHP
$a=1000;
echo "a=$a","<br>";
unset($a);
echo "a=$a"; //в данной строке будет ошибка, т.к. пере-
менной $a уже не существует
?>
```

Результаты выполнения последнего скрипта представлены на рис. 2.2.

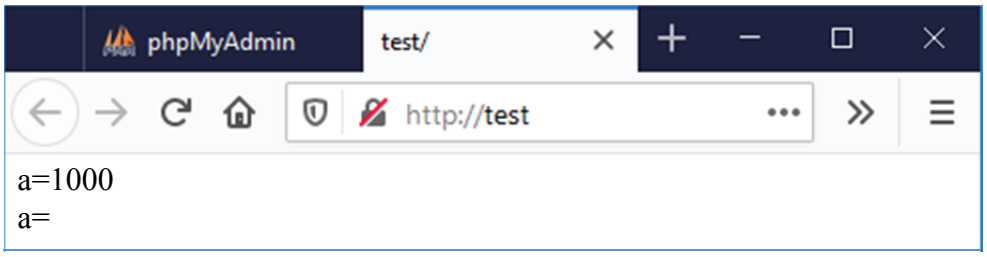

Рис. 2. 2. Пример создания и удаления переменных

Отличительным преимуществом PHP является то, что в нем не нужно ни описывать переменные явно, ни указывать их тип. Интерпретатор все это делает сам. Однако иногда он может ошибаться (например, если в текстовой строке на самом деле задано десятичное число), поэтому изредка возникает необходимость явно указывать, какой же тип имеет то или иное выражение. Чуть чаще возникает потребность узнать тип переменной (например, переданной в параметрах функции) прямо во время выполнения программы.

2.2. Типы данных (переменных) в PHP

PHP поддерживает восемь простых типов данных (переменных):

1. Четыре скалярных типа:
− boolean (двоичные данные);
− integer (целые числа);
− float (числа с плавающей точкой или 'double');
− string (строки).
2. Два смешанных типа:
− array (массивы);
− object (объекты).
3. Два специальных типа:
− resource (ресурсы);
− NULL («пустой» тип).

Существуют также несколько псевдотипов:
а) mixed (смешанный);
б) number (числовой);
в) callback (обратного вызова).

Рассмотрим кратко перечисленные типы данных PHP.

2.2.1. Тип boolean. Простейший тип, который выражает истинность значения. Это может быть либо TRUE, либо FALSE. Для определения булева тип используют ключевое слово TRUE или FALSE, причем оба являются регистронезависимыми.

```
<?PHP
   $x = True; // присвоить $x значение TRUE
?>
```

Допускаются следующие приведения типов:

− (int), (integer) – приведение к integer;
− (bool), (boolean) – приведение к boolean;
− (float), (double), (real) – приведение к float;
− (string) – приведение к string;
− (array) – приведение к array;
− (object) – приведение к object;
− (unset) – приведение к NULL.

Обратите внимание, что внутри скобок допускаются пробелы и символы табуляции, поэтому следующие примеры равносильны по своему действию.

```
<?PHP
   $foo = (int) $bar;
   $foo = ( int ) $bar;
?>
```

Для преобразования значения в булев тип используют приведение типа (bool) или (boolean). Однако в большинстве случаев нет необходимости применять приведение типа, поскольку значение будет автоматически преобразовано, если оператор, функция или управляющая конструкция требуют булев аргумент.

При преобразовании в логический тип следующие значения рассматриваются как FALSE:

− сам булев FALSE:
− целое 0 (ноль);
− число с плавающей точкой 0.0 (ноль);
− пустая строка и строка "0";
− массив с нулевыми элементами;
− объект с нулевыми переменными-членами;
− специальный тип NULL (включая неустановленные переменные).

Все остальные значения рассматриваются как TRUE (включая любой ресурс).

Также необходимо обратить внимание, что «–1» считается TRUE, как и любое ненулевое (отрицательное или положительное) число.

```
<?PHP
   var_dump((bool) "");        // bool(false)
   var_dump((bool) 1);         // bool(true)
   var_dump((bool) -2);        // bool(true)
   var_dump((bool) "foo");     // bool(true)
   var_dump((bool) 2.3e5);     // bool(true)
   var_dump((bool) array(12)); // bool(true)
   var_dump((bool) array());   // bool(false)
   var_dump((bool) "false");   // bool(true)
?>
```

Функция *var_dump ()* выводит в браузер структурированную информацию о переменной, а именно тип и значение. Например, выполнение следующего кода дает вот такой результат. Данная функция полезна для понимания процесса выполнения кода.

```
<?PHP
   $a=5;
   var_dump($a);
?>
```

Итоги выполнения скрипта представлены на рис. 2.3.

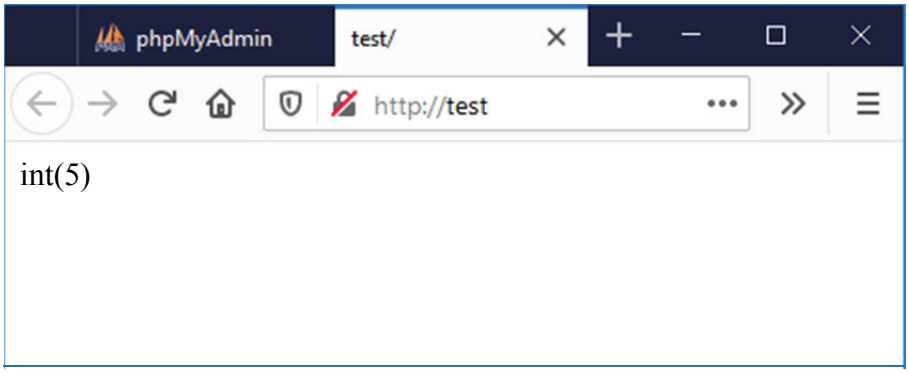

Рис. 2.3. Пример использования функции var_dump

В результате выполнения скрипта выведено сообщение о типе переменной и значения.

2.2.2. Тип integer. Целое – это число из множества Z = = {..., –2, –1, 0, 1, 2, ...} обычно длиной 32 бита (от –2 147 483 648 до 2 147 483 647).

Целые могут быть указаны в десятичной, шестнадцатеричной или восьмеричной системе счисления, по желанию с предшествующим знаком «–» или «+».

При использовании восьмеричной системы счисления число нужно предварить 0 (нулем), для использования шестнадцатеричной системы следует поставить перед числом 0x.

```
<?PHP
   $a = -1234; // отрицательное число
   $a = 123; // десятичное число
   echo "a=$a", "<br>";
   $a = 0123; // восьмеричное число (эквивалентно 83 в
десятичной системе)
   echo "переменная a, введенная в восьмеричной системе
счисления =$a","<br>";
   $a = 0x1A; // шестнадцатеричное число (эквивалентно
26 в десятичной системе)
   echo "переменная a, введенная в шестнадцатеричной
системе счисления =$a","<br>";
?>
```

Результаты выполнения скрипта представлены на рис. 2.4.

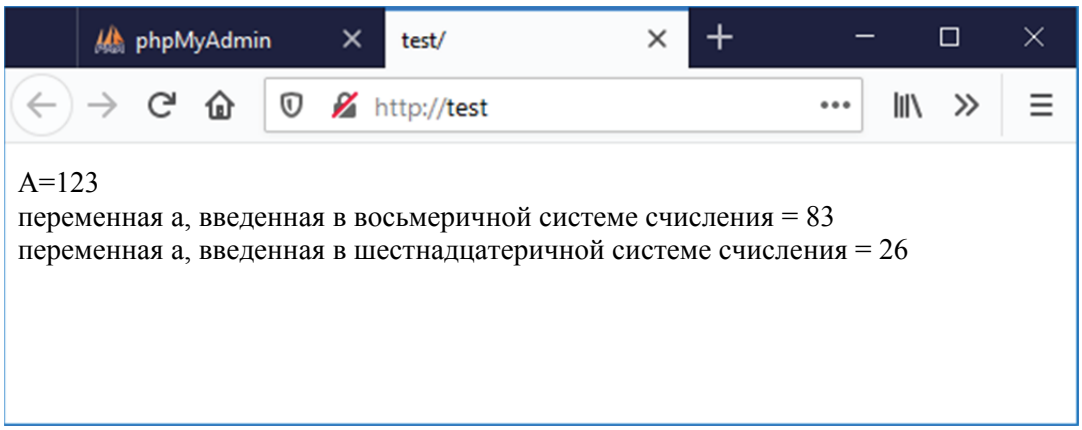

Рис. 2.4. Пример использования различных методов ввода чисел типа integer

Формально возможная следующая структура целых представлена на рис. 2.5.

```
десятичные   :  [1-9] [0-9] *
шестнадцатеричные:  0[xX] [0-9a-fA-F] +
восьмеричные:  0[0-7] +
целые:
[+-]?десятичные
|   [+-] ?шестнадцатеричные
|   [+-] ?восьмеричные
```

Рис. 2.5. Пример структуры целых чисел

Размер целого зависит от платформы, хотя, как правило, максимальное значение около двух миллиардов (это 32-битное знаковое). PHP не поддерживает беззнаковые (начинающиеся от нуля) целые.

Если определить число, превышающее пределы целого типа, оно будет интерпретировано как число с плавающей точкой, т. е. ошибка не возникнет. Также, если использовать оператор, результатом работы которого будет число, превышающее пределы целого, вместо него будет возвращено число с плавающей точкой.

```
<?PHP
$large_number =  2147483647;
var_dump($large_number); // вывод: int(2147483647)

$large_number =  2147483648;
var_dump($large_number); // вывод: float(2147483648)

// это справедливо и для шестнадцатеричных целых:
var_dump( 0x80000000 ); // вывод: float(2147483648)

$million = 1000000;
$large_number =  50000 * $million;
var_dump($large_number); // вывод: float(50000000000)
?>
```

В PHP не существует оператора деления целых. Результатом ½ будет число с плавающей точкой 0,5. Вы можете привести значение к целому, что всегда округляет его в меньшую сторону, либо использовать функцию round(), округляющую по традиционным правилам математики. Однако, если в результате деления двух чисел предполагается целое число, например 25 / 5 = 5, то тип данных действительно получится integer.

```
<?PHP
    $a=25/7;
    var_dump($a);// float(3.5714285714286)
?>
<br>
<?PHP
    $b=(int) $a;
    var_dump($b); // int(3)
?>
<br>
<?PHP
    $c=round($a);
    var_dump($c); //float (4)
?>
<br>
<?PHP
    $a=25/5;
    var_dump($a);// int(5)
?>
```

Результаты выполнения скрипта представлены на рис. 2.6.

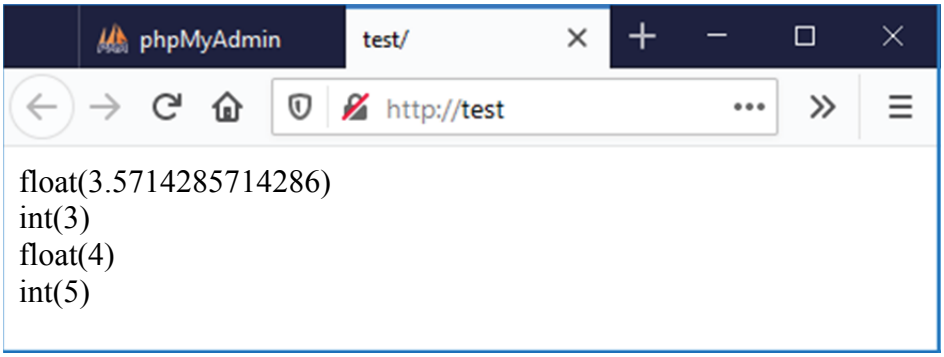

Рис. 2.6. Результаты математических операций
с переменными типа integer

Для однозначного преобразования значения в целое используют приведение типа (int) или (integer). Однако в большинстве случаев нет необходимости применять приведение типа, поскольку значение будет автоматически преобразовано, если оператор, функция или управляющая конструкция требуют целый аргумент. Можно также преобразовать значение в целое при помощи функции *intval()*.

При преобразовании из числа с плавающей точкой (float) в целое число будет округлено в меньшую сторону.

```
<?PHP
    $a=1.55;
    $b=intval($a);//преобразование типа float в integer
    Var_dump($b);// b будет равно 1
?>
```

Результаты выполнения скрипта представлены на рис. 2.7.

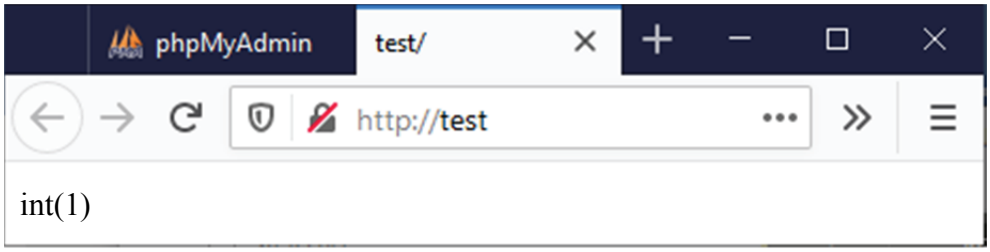

Рис. 2.7. Пример использования функции intval()
для преобразования в integer

Если число с плавающей точкой превышает пределы целого (как правило, это +/– $2.15e+9 = 2^{31}$), результат будет неопределенным, так как целое не имеет достаточной точности, чтобы вернуть верный результат. В этом случае не будет выведено ни предупреждения, ни даже замечания! Поэтому никогда не нужно приводить неизвестную дробь к целому, так как это может дать неожиданные результаты. Пример ошибочного преобразования представлен в следующем примере.

```
<?PHP
    echo (int) ((0.1+0.7) * 10); // выводит 7! (ошиб-
ка!)
?>
```

Результаты выполнения скрипта приведены на рис. 2.8.

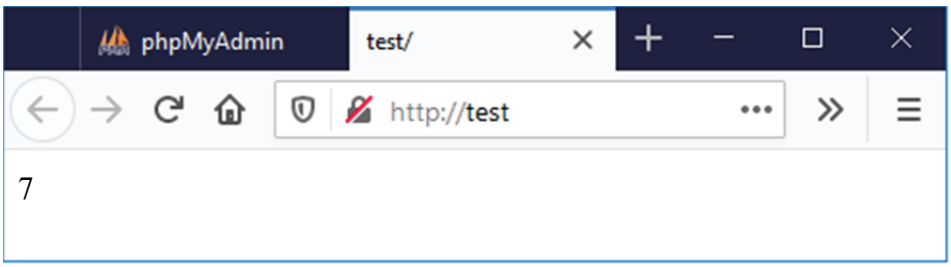

Рис. 2.8. Пример ошибочного преобразования дроби в integer

Такой результат получается из-за того, что потенциальные 0.8, полученные при сложении 0,1 + 0,7, ввиду особенностей хранения в памяти (невозможностью точно выразить некоторые дроби в десятичной системе счисления конечным числом цифр) будут представлены в виде бесконечной дроби 0,7(9), а при применении операции *int()* результатом будет 7 как целое число.

При преобразовании переменных типа boolean в integer FALSE преобразуется в 0 (ноль), а TRUE – в 1 (единицу).

```
<?PHP
    $a=false;
    $b=intval($a);//преобразование типа boolean в inte-
ger.    Результат будет равен 0
    echo $b;
?>
```

Результаты выполнения скрипта представлены на рис. 2.9.

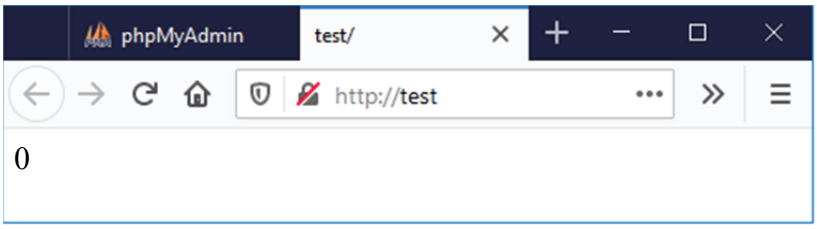

Рис. 2.9. Пример преобразования boolean в integer

Целые числа широко используются при написании программ на языке PHP, например для выполнения расчетов или в качестве переменных циклов.

2.2.3. Тип float (или double). Тип float (числа с плавающей точкой) – вещественное число довольно большой точности (ее должно хватить для подавляющего большинства математических вычислений). Числа с плавающей точкой (они же числа двойной точности, или действительные числа) могут быть определены при помощи любого из представленных синтаксисов:

```
<?PHP
    $a = 1.234;
    $b = 1.2e3;
    $c = 7E-10;
?>
```

Формальная структура скалярного типа float представлена следующим образом (рис. 2.10):

LNUM	[0–9] +	
DNUM	([0–9] * [.]{LNUM})	({LNUM} [.] [0–9] *)
EXPONENT_DNUM	(({LNUM} | {DNUM})	[eE] [+–] ? {LNUM})

Рис. 2.10. Формальная структура скалярного типа float

Размер переменной типа float зависит от платформы, хотя максимум, как правило, ~$1.8 \cdot 10^{308}$ с точностью около 14 десятичных цифр (это 64-битный IEEE-формат).

Довольно часто простые десятичные дроби вроде 0,1 или 0,7 не могут быть преобразованы в свои внутренние двоичные аналоги без небольшой потери точности. Это может привести к неожиданным результатам: например, floor((0.1 + 0.7)*10) скорее всего возвратит 7 вместо ожидаемой 8 как результат внутреннего представления числа, являющегося в действительности чем-то вроде 7.9999999999.... (floor – округляет дробь в меньшую сторону).

```
<?PHP
    $a=floor((0.1+0.7)*10);
    echo $a;// b будет равно 7
?>
```

Результаты выполнения скрипта представлены на рис. 2.11.

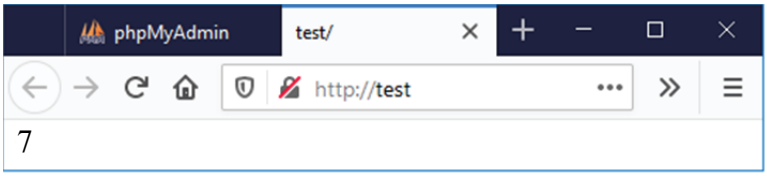

Рис. 2.11. Пример некорректной работы с дробями

Это связано, как уже отмечалось выше, с невозможностью точно выразить некоторые дроби в десятичной системе счисления конечным числом цифр. Например, 1/3 в десятичной форме принимает вид 0.3333333... .

Никогда нельзя доверять точности последних цифр в результатах с числами с плавающей точкой и никогда не нужно проверять их на равенство. Если вам действительно необходима высокая точность, то следует использовать математические функции произвольной точности или *gmp*-функции.

2.2.4. Тип string. Строка в PHP – это набор символов любой длины. В отличие от языка программирования C, строки могут содержать в себе также и нулевые символы, что никак не повлияет на программу. Иными словами, строки можно использовать для хранения бинарных данных. Длина строки ограничена только размером свободной оперативной памяти.

Строка может быть обработана при помощи стандартных функций, а можно непосредственно обратиться к любому ее символу (подробнее рассмотрим в гл. 3).

Приведем простейший пример использования строковой переменной (ввода и вывода).

```
<?PHP
    $a = "Это просто текст, записанный в строковую переменную";
    echo $a; //Выводит 'Это просто текст, записанный в строковую переменную'
?>
```

Результаты выполнения скрипта представлены на рис. 2.12.

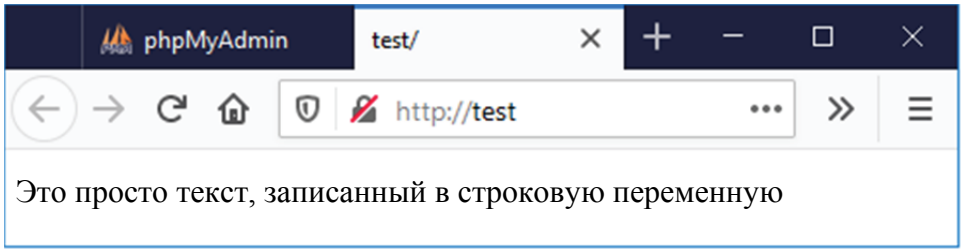

Рис. 2.12. Пример работы с переменными string

А теперь подробно разберем синтаксис типа данных string. Строка может быть определена тремя различными способами.

− одинарными кавычками;
− двойными кавычками;
− heredoc-синтаксисом;
− nowdoc-синтаксисом.

Простейший способ определить строку – это заключить ее в одинарные кавычки (' ').

Для использования одинарной кавычки внутри строки, как и во многих других языках, ее необходимо предварить символом обратной косой черты (\), т. е. экранировать ее. Если обратная косая

черта должна идти перед одинарной кавычкой либо быть в конце строки, необходимо продублировать ее. Обратите внимание, что при попытке экранировать любой другой символ, обратная косая черта также будет выведена на экран. Так что, как правило, нет необходимости экранировать саму обратную косую черту.

В отличие от двух других синтаксисов, переменные и экранирующие последовательности для специальных символов, встречающиеся в строках, заключенных в одинарные кавычки, не обрабатываются. Рассмотрим несколько примеров использования одинарных кавычек.

```
<?PHP
    echo '<p>';
    echo 'Сегодня отличная погода!';
    echo '</p>';
    echo '<p>';
    echo 'Сёння выдатнае надвор\'е!';
    echo '</p>';
?>
```

Результаты выполнения скрипта представлены на рис. 2.13.

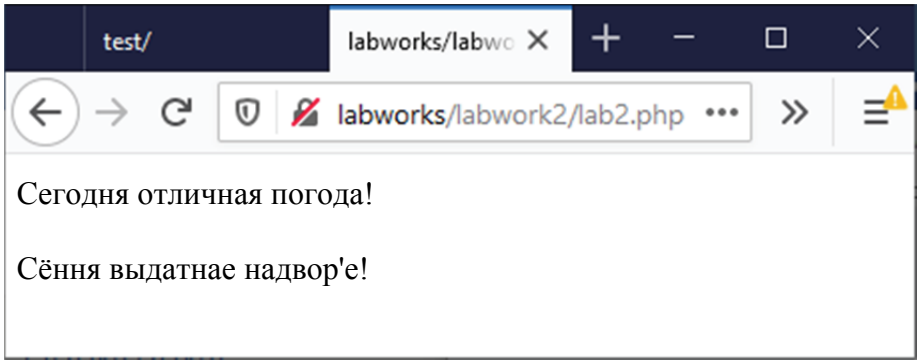

Рис. 2.13. Пример экранирования специальных символов в строке

Если строка заключена в двойные кавычки ("), PHP распознает большее количество управляющих последовательностей для специальных символов (представлены в таблице).

Таблица управляющих последовательностей

Последовательность	Значение
\n	Новая строка (LF или 0×0A (10) в ASCII)
\r	Возврат каретки (CR или 0×0D (13) в ASCII)

Окончание табл.

Последовательность	Значение
\t	Горизонтальная табуляция (HT или 0×09 (9) в ASCII)
**	Обратная косая черта
\$	Знак доллара
\"	Двойная кавычка
\[0-7]{1,3}	Последовательность символов, соответствующая регулярному выражению, символ в восьмеричной системе счисления
\x[0-9A-Fa-f]{1,2}	Последовательность символов, соответствующая регулярному выражению, символ в шестнадцатеричной системе счисления

Отметим, что если необходимо мнемнонизировать любой другой символ, обратная косая черта также будет напечатана! Самым важным свойством строк в двойных кавычках является обработка переменных. Например, данную особенность можно использовать при выводе значений переменных и т. д.

```
<?PHP
    $a = 2;
    $b=10;
    echo "Переменная a= $a , а переменная b= $b .";
//Выводит значения a и b, встроенные в текст
?>
```

Результаты выполнения скрипта представлены на рис. 2.14.

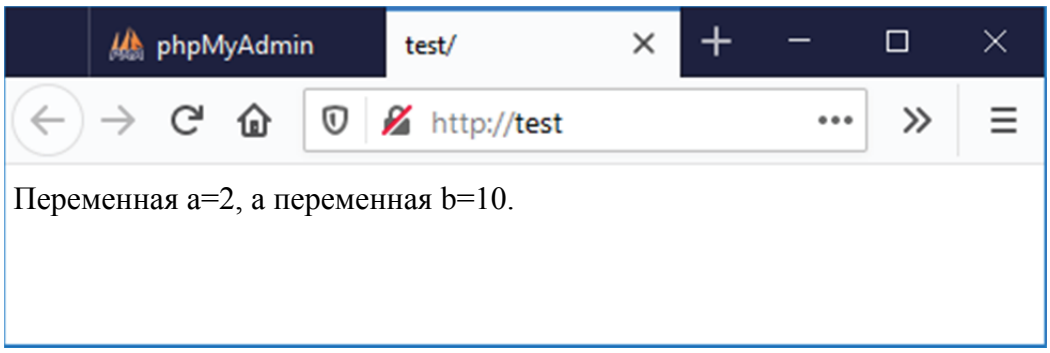

Рис. 2.14. Пример использования вывода текста и значений переменных оператором echo с помощью двойных кавычек

Еще одни способ определения строк – это использование heredoc-синтаксиса ("<<<"). После <<< необходимо указать идентификатор,

затем идет строка, а потом этот же идентификатор, закрывающий вставку.

Закрывающий идентификатор должен начинаться в первом столбце строки. Кроме того, идентификатор должен соответствовать тем же правилам именования, что и все остальные метки в PHP: содержать только буквенно-цифровые символы и знак подчеркивания и должен начинаться не с цифры или знака подчеркивания.

```
<?PHP
echo <<<TEXT
Hello world!
TEXT;
?>
```

Результаты выполнения скрипта представлены на рис. 2.15.

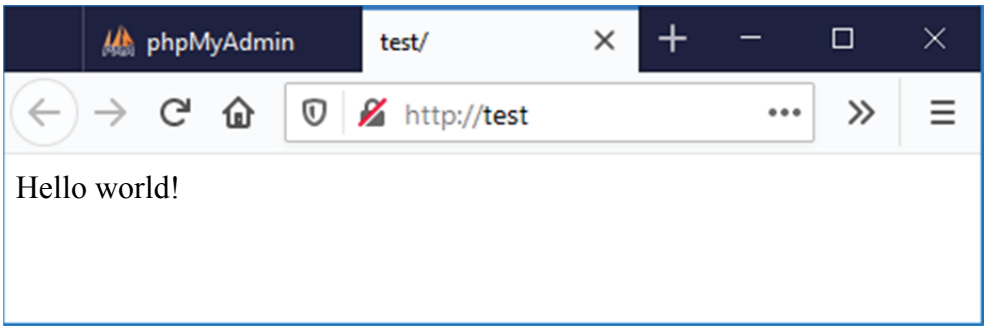

Рис. 2.15. Результаты использования heredoc-синтаксиса

Heredoc-текст ведет себя так же, как и строка в двойных кавычках, при этом их не имея. Это означает, что нет необходимости экранировать кавычки в heredoc, но по-прежнему можно использовать вышеперечисленные управляющие последовательности. Переменные обрабатываются, но с применением сложных переменных внутри heredoc нужно быть также внимательным, как и при работе со строками.

```
<?PHP
$a=10;
echo <<<TEXT
Значение переменной a=$a
TEXT;
?>
```

Результаты выполнения скрипта представлены на рис. 2.16.

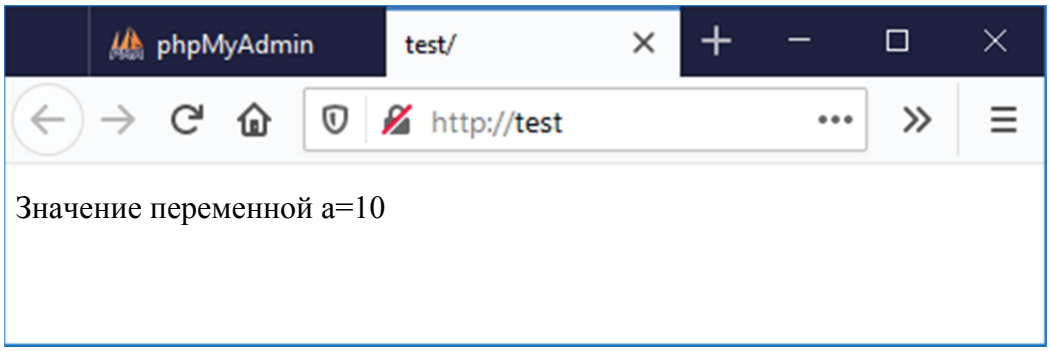

Рис. 2.16. Пример использования переменных при выводе текста с использованием heredoc-синтаксиса

Важно отметить, что строка с закрывающим идентификатором не содержит других символов, за исключением, возможно, точки с запятой (;). Это означает, что идентификатор не должен вводиться с отступом и не может быть никаких пробелов или знаков табуляции до либо после точки с запятой. Важно также понимать, что первым символом перед закрывающим идентификатором должен быть символ новой строки, определенный в операционной системе. Например, для Windows – это \r. Если это правило нарушено и закрывающий идентификатор не является «чистым», считается, что закрывающий идентификатор отсутствует и PHP продолжит его поиск дальше. Если в этом случае верный закрывающий идентификатор так и не будет найден, то это вызовет ошибку в обработке с номером строки в конце скрипта.

Nowdoc − это то же самое для строк в одинарных кавычках, что и heredoc для строк в двойных кавычках. Nowdoc похож на heredoc, но внутри него *не осуществляется никаких подстановок*. Данная конструкция идеальна для внедрения PHP-кода или других больших блоков текста без необходимости его экранирования. В этом он немного похож на SGML-конструкцию *<![CDATA[ ]]>* тем, что объявляет блок текста, не предназначенный для обработки.

Nowdoc указывается той же последовательностью <<<, что используется в heredoc, но последующий за ней идентификатор заключается в одинарные кавычки, например *<<<'EOT'*. Все условия, действующие для идентификаторов heredoc, также действительны и для nowdoc, особенно те, что относятся к закрывающему идентификатору.

```
<?PHP
echo <<<'EOD'
    Пример текста,
    занимающего несколько строк,
    с помощью синтаксиса nowdoc.
EOD;
?>
```

Результаты выполнения скрипта представлены на рис. 2.17.

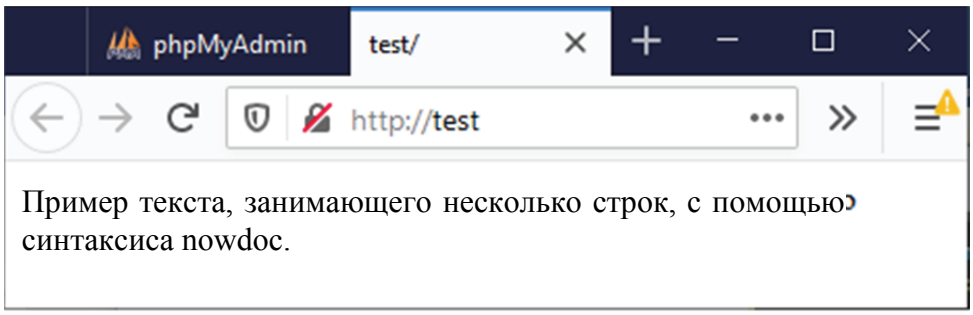

Рис. 2.17. Результаты использования nowdoc-синтаксиса

2.2.5. Тип array. Массив в PHP – это упорядоченный набор данных, в котором установлено соответствие между значением и ключом. Индекс (ключ) служит для однозначной идентификации элемента внутри массива. В одном массиве не может быть двух элементов с одинаковыми индексами.

PHP позволяет создавать массивы любой сложности.

1. *Простой массив (список).* Массивы, индексами которых являются числа, начинающиеся с нуля, обычно называются списками.

```
<?PHP
    // Простой способ инициализации массива
    $names[0]="Апельсин";
    $names[1]="Банан";
    $names[2]="Груша";
    $names[3]="Помидор";
    // Здесь: names - имя массива, а 0, 1, 2, 3 - ин-
дексы массива
    echo $names[1]; // Выведет на экран Банан
?>
```

Результаты выполнения скрипта представлены на рис. 2.18.

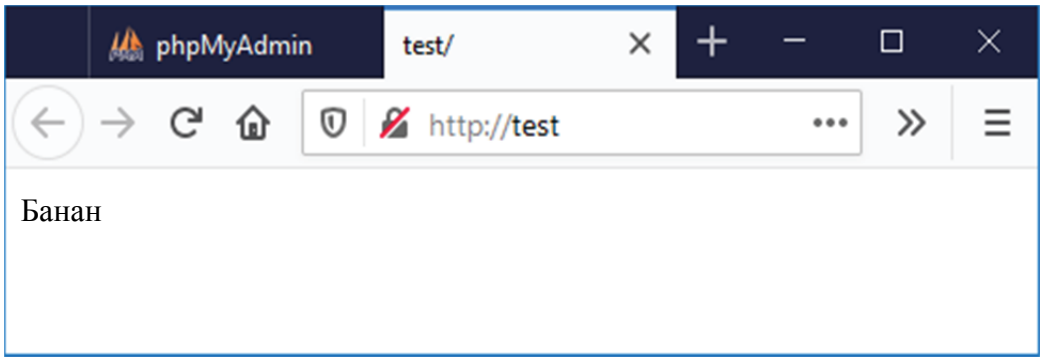

Рис. 2.18. Пример использования списков

Отметим, что нумерация в простом массиве может начинаться и не с нуля.

```
<?PHP
    // Простой способ инициализации массива
    $names[1]="Апельсин";
    $names[2]="Банан";
    $names[3]="Груша";
    $names[4]="Помидор";
    // Здесь: names - имя массива, а 0, 1, 2, 3 - ин-
дексы массива
    echo $names[1]; // Выведет на экран Апельсин
?>
```

Результаты выполнения скрипта представлены на рис. 2.19.

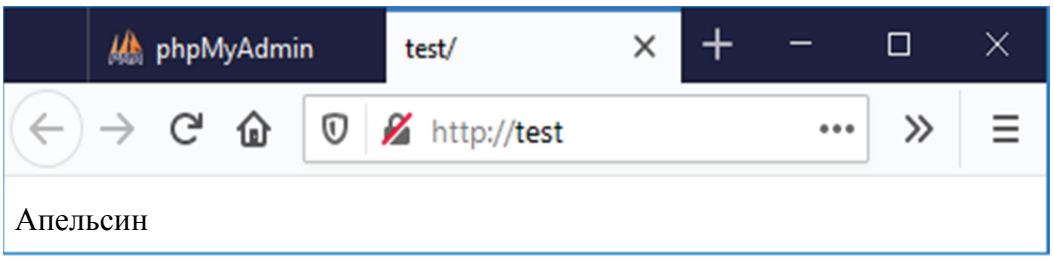

Рис. 2.19. Пример использования списков
с началом нумерации, отличной от 0

Доступ к элементам простых массивов (списков), как показано в предыдущих примерах, осуществляется по номеру элемента в простом массиве (списке).

Простые массивы можно создавать, не указывая индекс нового элемента массива, это за вас сделает PHP.

```
<?PHP
    // Простой способ инициализации массива, без указа-
ния индексов
    $names[]="Апельсин";
    $names[]="Банан";
    $names[]="Груша";
    $names[]="Помидор";
    // PHP автоматически присвоит индексы элементам
массива, начиная с 0

    // Выводим элементы массивов в браузер:
    echo $names[0]; // Вывод элемента массива names с
индексом 0, т. е. апельсин
    echo "<br>";
    echo $names[3]; // Вывод элемента массива names с
индексом 3, т. е. помидор
?>
```

Результаты выполнения скрипта представлены на рис. 2.20.

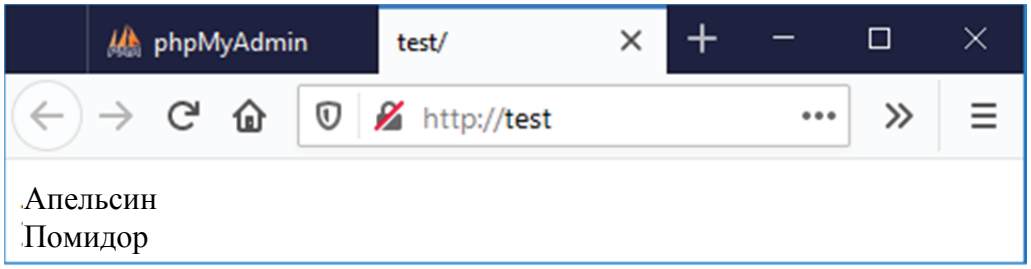

Рис. 2.20. Пример использования списков
(при вводе не указывался индекс)

В рассмотренном примере можно добавлять элементы массива names простым способом, т. е. не указывая индекс элемента массива.

```
$names[]="Яблоко";
```

Новый элемент простого массива (списка) будет добавлен в конец массива. В дальнейшем с каждым новым элементом массива индекс будет увеличиваться на единицу.

2. *Простые многомерные массивы.* Обобщенный синтаксис элементов многомерного простого массива.

```
$имя[индекс1][индекс2]..[индексN];
```

Примером простого многомерного массива может быть массив 3×3 (нумерация начинается с 0).

```
<?PHP
    // Многомерный простой массив:
    $arr[0][0]="Овощи";
    $arr[0][1]="Фрукты";
    $arr[1][0]="Абрикос";
    $arr[1][1]="Апельсин";
    $arr[1][2]="Банан";
    $arr[2][0]="Огурец";
    $arr[2][1]="Помидор";
    $arr[2][2]="Тыква";

    // Выводим некоторые элементы массива:
    echo "<p>".$arr[1][0]."</p>";
    echo "<p>".$arr[2][1]."</p>";
?>
```

Результаты выполнения скрипта представлены на рис. 2.21.

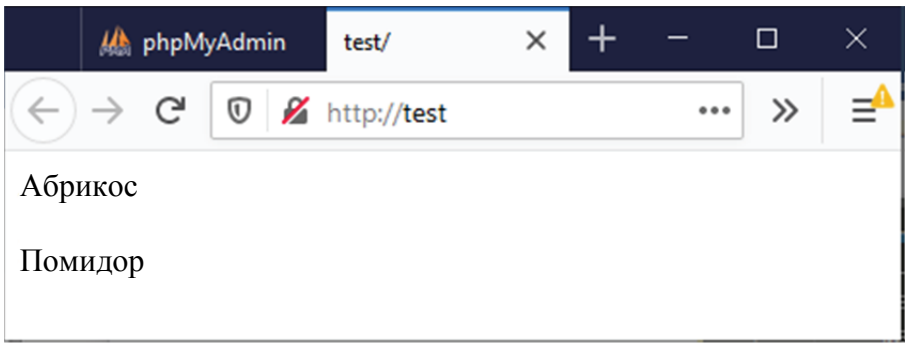

Рис. 2.21. Пример использования многомерного массива

В последнем примере применена операция конкатенации (объединения строк), которая обозначается как «.».

3. *Ассоциативные массивы.* В PHP индексом массива может быть не только число, но и строка. Причем на такую строку не накладываются никакие ограничения: она может содержать пробелы, длина такой строки может быть любой.

Ассоциативные массивы особенно удобны в ситуациях, в которых элементы массива нужно связывать со словами, а не с числами.

Итак, массивы, индексами которых являются строки, называются ассоциативными массивами, они бывают двух типов: одномерные и многомерные.

Одномерные ассоциативные массивы содержат только один ключ (элемент), соответствующий конкретному индексу ассоциативного массива.

```
<?PHP

    // Ассоциативный массив
    $names["Иванов"]="Иван";
    $names["Сидоров"]="Николай";
    $names["Петров"]="Петр";
    // В данном примере: фамилии - ключи ассоциативного
массива, а имена - элементы массива
    echo $names["Сидоров"]; //выводит имя Николай

?>
```

Результаты выполнения скрипта представлены на рис. 2.22.

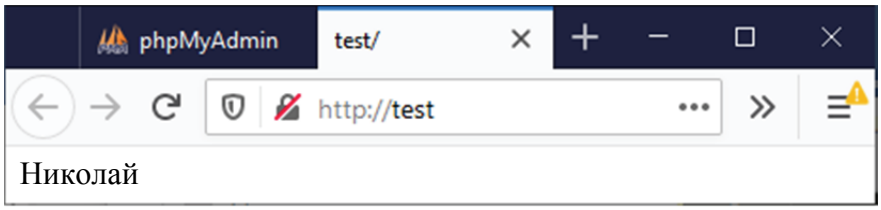

Рис. 2.22. Пример использования ассоциативного массива

Доступ к элементам одномерных ассоциативных массивов, как видно из предыдущего примера, осуществляется так же, как и к элементам обыкновенных массивов, и называется доступом по ключу.

```
echo $names["Сидоров"]; //выводит имя Николай
```

Многомерные ассоциативные массивы могут содержать несколько ключей, соответствующих конкретному индексу ассоциативного массива. Пример многомерного ассоциативного массива представлен на рис. 2.23.

	name	age	email
Ivanov	Иванов И. И.	25	ivanov@mail.ru
Petrov	Петров П. П.	34	petrov@mail.ru
Sidorov	Сидоров С. С.	47	sidorov@mail.ru

Рис. 2.23. Пример многомерного ассоциативного массива

На PHP работа с таким многомерным ассоциативным массивом может выглядеть следующим образом:

```
<?PHP
    // Многомерный массив
    $A["Ivanov"] = array("name"=>"Иванов И.И.",
"age"=>"25", "email"=>"ivanov@mail.ru");
    $A["Petrov"] = array("name"=>"Петров П.П.",
"age"=>"34", "email"=>"petrov@mail.ru");
    $A["Sidorov"] = array("name"=>"Сидоров С.С.",
"age"=>"47", "email"=>"sidorov@mail.ru");

    echo $A["Petrov"]["name"];// Выводит Иванов И.И
    echo "<br>";
    echo $A["Petrov"]["age"];// Выводит возраст
    echo "<br>";
    echo $A["Petrov"]["email"];// Выводит e-mail
?>
```

Результаты выполнения скрипта представлены на рис. 2.24.

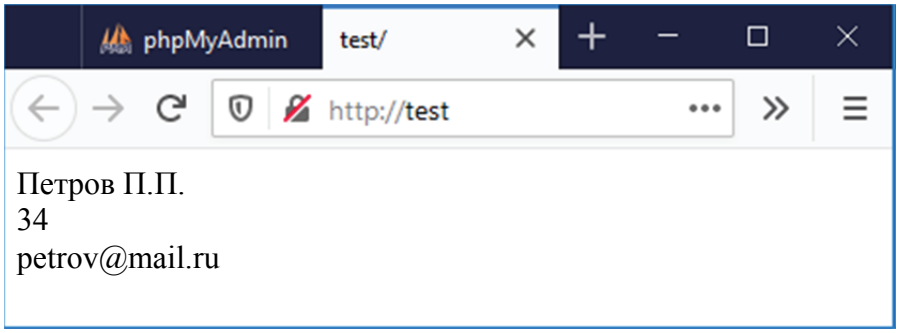

Рис. 2.24. Пример использования
многомерного ассоциативного массива

Доступ к элементам многомерного ассоциативного массива осуществляется через указание двух ключей:

```
echo $A["Petrov"]["name"];// Выводит Иванов И.И
echo $A["Petrov"]["age"];// Выводит возраст
echo $A["Petrov"]["email"];// Выводит e-mail
```

Отметим, что для создания многомерного ассоциативного массива была использована специальная функция *array*. Ассоциативные многомерные массивы можно создавать и классическим способом, хотя это не так удобно.

```
<?PHP
    // Многомерный ассоциативный массив
    $A["Ivanov"]["name"]="Иванов И.И.";
    $A["Ivanov"]["age"]="25";
    $A["Ivanov"]["email"]="ivanov@mail.ru";

    $A["Petrov"]["name"]="Петров П.П.";
    $A["Petrov"]["age"]="34";
    $A["Petrov"]["email"]="petrov@mail.ru";

    $A["Sidorov"]["name"]="Сидоров С.С.";
    $A["Sidorov"]["age"]="47";
    $A["Sidorov"]["email"]="sidorov@mail.ru";

    //Получаем доступ к ключам многомерного ассоциатив-
ного массива
    echo $A["Petrov"]["name"];// Выводит Иванов И.И
    echo "<br>";
    echo $A["Petrov"]["age"];// Выводит возраст
    echo "<br>";
    echo $A["Petrov"]["email"];// Выводит e-mail
?>
```

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

Для инициализации объекта применяется выражение new, создающее в переменной экземпляр объекта.

```
<?PHP
    class ob
    {
        function do_ob()
        {
            echo "I use object.";
        }
    }
    $n_ob = new ob;
    $n_ob->do_ob();
?>
```

Результаты выполнения скрипта представлены на рис. 2.25.

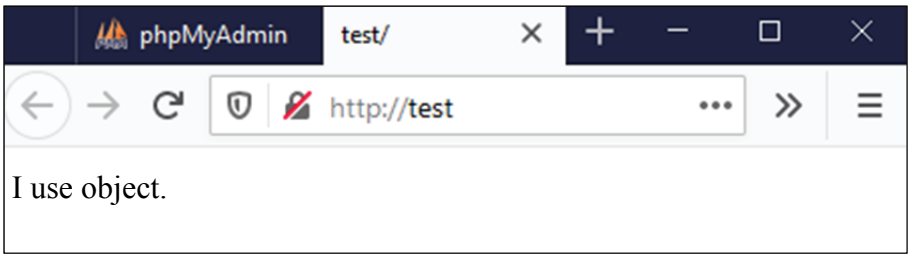

Рис. 2.25. Пример использования типа object

Более подробное рассмотрение объектов будет производиться в теме, посвященной объектно-ориентированному программированию на языке PHP.

2.2.7. Тип resource. Ресурс – это специальная переменная, содержащая ссылку на внешний ресурс. Ресурсы создаются и используются специальными функциями.

Тип ресурс содержит специальные указатели: на открытые файлы, соединения с базой данных, область изображения и т. п., поэтому нельзя преобразовать какое-либо значение в ресурс.

В связи с системой подсчета ссылок, введенной в движке Zend, автоматически определяется, что ресурс больше никуда не ссылается (как в Java). Когда это происходит, все ресурсы, используемые для данного ресурса, освобождаются. По этой причине маловероятно, что когда-либо будет необходимо освобождать память вручную с помощью free_result функции.

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

Переменная считается NULL если:

− ей была присвоена константа NULL;

− еще не было присвоено какое-либо значение;

− она была удалена с помощью unset().

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

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

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

Псевдотип number (числовой) говорит о том, что параметр может быть либо integer, либо float.

Псевдотип callback (обратного вызова) предполагает, что некоторые функции, такие как call_user_func() или usort() принимают в качестве параметра определенные пользователем callback-функции. Callback-функции могут быть не только простыми функциями, но также методами объектов, включая статические методы классов. PHP-функция передается просто как строка ее имени. Вы можете передать любую встроенную или определенную пользователем функцию за исключением array(), echo(), empty(), eval(), exit(), isset(), list(), print() и unset().

2.3. Таблицы сравнения типов

Для корректного понимания языка PHP важно четко знать процесс сравнения переменных раличных типов. Зачастую в ходе работы скрипта необходимо определять тип переменной, например числовая или нет, а также четко установить факт существования переменной. Для этих целей используются функции gettype(), empty(), is_null(), isset(), is_numeric(). HTML-формы не передают тип переменной, они всегда передают строки. Для проверки, является ли строка числом, используйте функцию is_numeric(). Применение if ($x), пока $x не определена, сгенерирует ошибку E_NOTICE. Вместо этого надо использовать функцию empty() или isset() и/или инициализировать переменную.

Таблицы, представленные в прил. А, демонстрируют работу PHP с типами переменных и операторами сравнения как в случае свободного, так и в случае строгого сравнения.

Практическая часть

Пример 1

Рассмотрим пример создания трех переменных, которые называются *зима* и содержат строку, целое число и вещественное

число. Выведем значения этих переменных и их тип в окне браузера. Для определения типа значений используем функцию gettype(). Для вывода имени переменной перед знаком доллара пишем символ бек-слеш «\» (более подробно будем рассматривать в гл. 5). Этот же знак используем в последней функции echo, чтобы вывести текст «Тень-2» в двойных кавычках. Код сохраним в файле plan_ten.php.

```
<?PHP
  $Zima = "-1";
  $zima = -1;
  $_Zima = -1.0;

  $type_Zima = gettype($Zima);
  $type_zima = gettype($zima);
  $type__Zima = gettype($_Zima);

  echo "Значение переменнной \$Zima = $Zima. Это тип
данных \"$type_Zima\"<br />";
  echo "Значение переменнной \$zima = $zima. Это тип
данных \"$type_zima\"<br />";
  echo "Значение переменнной \$_Zima = $_Zima. Это тип
данных \"$type__Zima\"<br />";
  echo "План \"Тень-2\" начнется через два-три меся-
ца<br />";</p>
```

Результаты выполнения скрипта представлены на рис. 2.26.

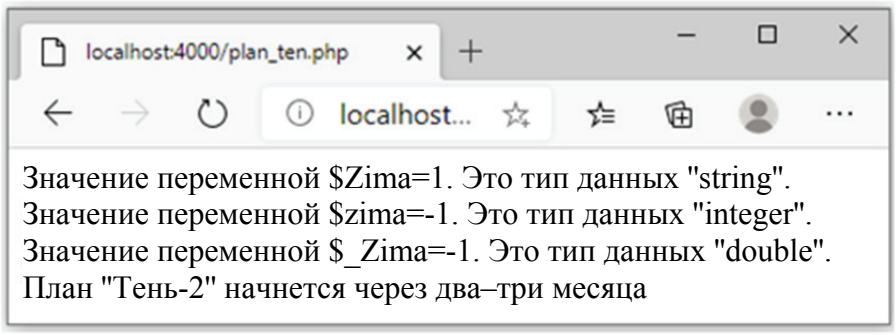

Рис. 2.26. Пример использования PHP и HTML

Пример 2

В данном примере выводятся максимальное и минимальное значения целых чисел. Числа зависят от типа операционной системы и версии PHP. Для определения этих значений воспользуемся

встроенными константами PHP_INT_MAX и PHP_INT_MIN. Информацию о версии PHP нам поможет получить функция PHPinfo(). Код сохраним в файле integer.php.

```
<?PHP
  $int_max=PHP_INT_MAX;
  echo "<br />Max integer: $int_max";
  $int_min=PHP_INT_MIN;
  echo "<br />Min integer: $int_min";
  $int_max32 = (2**32)/2-1;
  echo "<br />Max integer: $int_max32";
  PHPinfo();
?>
```

Результат выполнения данного скрипта в Windows 10×64 приведен на рис. 2.27.

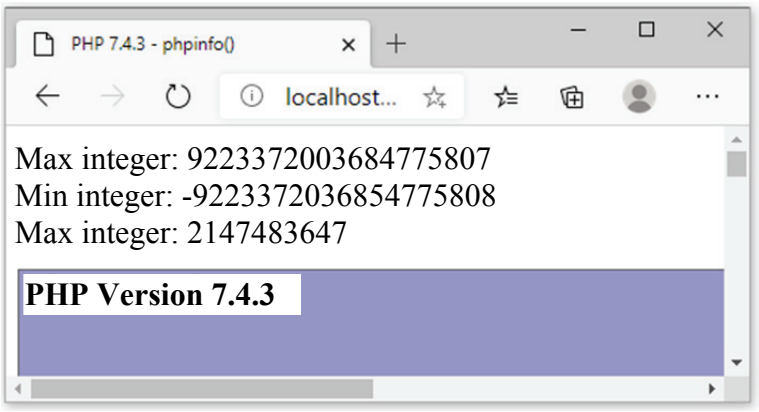

Рис. 2.27. Диапазон целых чисел

Выполните скрипт и проанализируйте полученные результаты для своей системы.

Пример 3

Рассмотрим несколько примеров использования одинарных кавычек строк и применения экранирования.

```
<?PHP
    echo 'это простая строка';
    echo "<br>","<br>";
    echo 'Также вы можете вставлять в строки
    символ новой строки таким образом,
```

```
    поскольку это нормально';
    echo "<br>","<br>";

    // Выведет: Однажды Арнольд сказал: "I'll be back"
    echo 'Однажды он сказал: "I\'ll be back"';
    echo "<br>","<br>";

    // Выведет: Вы удалили C:\*.*?
    echo 'Вы удалили C:\\*.*?';
    echo "<br>","<br>";

    // Выведет: Вы удалили C:\*.*?
    echo 'Вы удалили C:\*.*?';
    echo "<br>","<br>";

    // Выведет: Это не вставит: \n новую строку
    echo 'Это не вставит: \n новую строку';
    echo "<br>","<br>";
    // Выведет: Переменные $expand также $either не
подставляются
    echo 'Переменные $expand также $either не подстав-
ляются';
?>
```

Результаты выполнения скрипта представлены на рис. 2.28.

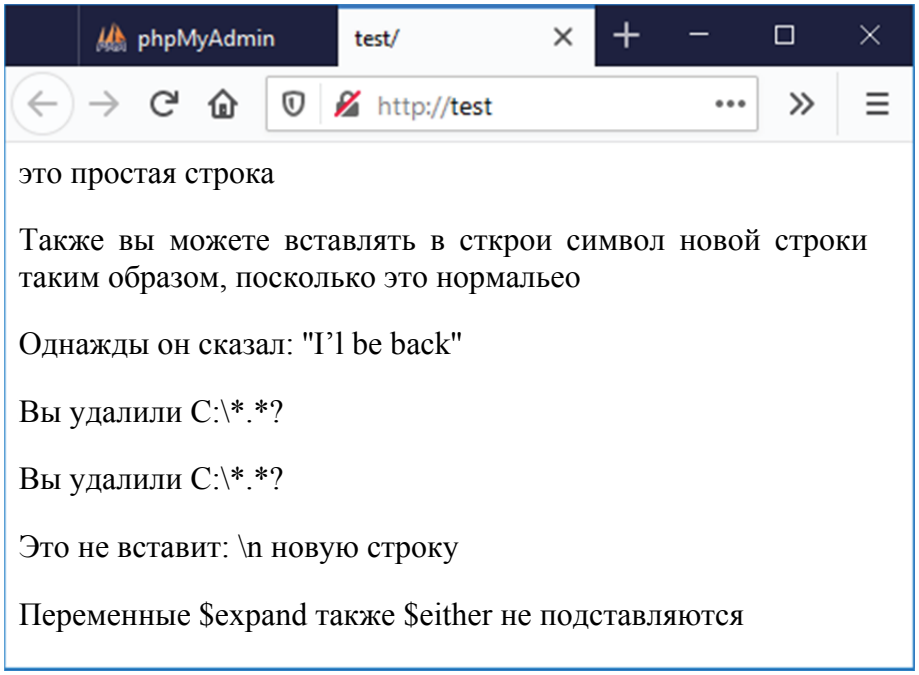

Рис. 2.28. Пример экранирования специальных символов в строке

Рассмотренные примеры показывают, что, используя экранирование специальных символов в строке, можно угадать формат выводимых результатов и управлять видом как численных, так и сроковых значений.

Лабораторная работа № 2

Задания на лабораторную работу будут носить похожий на рассмотренные примеры характер и выдаваться индивидульно преподавателем. Напомним, что выполнение и демострация всех лабораторных работ предполагается в пределах структуры, представленной в лабораторной работе № 1.

ГЛАВА 3

РАБОТА С ПЕРЕМЕННЫМИ

Для осуществления операций с переменными существуют различные группы операторов. Оператором называется нечто, состоящее из одного или более значений (выражений), которое можно вычислить как новое значение (таким образом, вся конструкция может рассматриваться как выражение). Отсюда следует, что функции или любые другие конструкции, которые возвращают значение (например, print()), являются операторами, в отличие от всех остальных языковых конструкций (например, echo()), которые ничего не возвращают.

3.1. Арифметические операции

Описанные в табл. 3.1 операторы работают в соответствии с логикой стандартных математических операций.

Таблица 3.1

Таблица арифметических операций

Пример	Название	Результат
–$*a*	Отрицание	Смена знака $*a*
$*a* + $*b*	Сложение	Сумма $*a* и $*b*
$*a* – $*b*	Вычитание	Разность $*a* и $*b*
$*a* * $*b*	Умножение	Произведение $*a* и $*b*
$*a* / $*b*	Деление	Частное от деления $*a* на $*b*
$*a* % $*b*	Деление по модулю	Целочисленный остаток от деления $*a* на $*b*
$*a* ** $*b*	Возведение в степень	Возводит значение переменной $*a* в степень, заданную значением переменной $*b*

Далее приведем простейший пример ввода двух переменных *a* и *b*, а также целочисленного деления *a* на *b* с выводом результата на экран.

```
<?PHP
  $a = 10;
  $b=4;
  $c=$a%$b;
  echo "Целочисленный остаток от деления a на b равен $c";
?>
```

Результаты выполнения скрипта представлены на рис. 3.1.

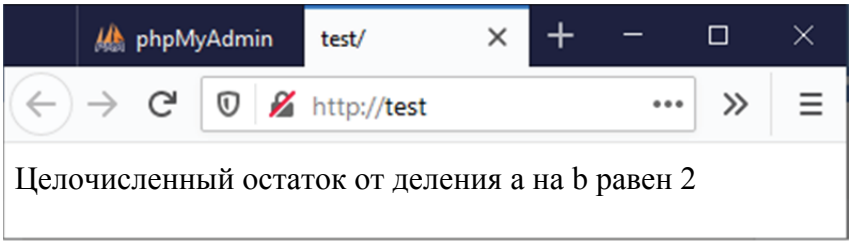

Рис. 3.1. Результат использования математических операций

Важно отметить, что операцию вычисления остатка от деления "%" рекомендуется использовать только с целыми числами, так как применение ее к дробным может привести к нежелательному и непредсказуемому результату.

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

3.2. Операции инкремента и декремента

Язык PHP поддерживает префиксные и постфиксные операторы инкремента и декремента, которые представлены в табл. 3.2.

Таблица 3.2

Таблица операций инкремента и декремента

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

Постфиксные операторы инкремента и декремента, как и в языке C, увеличивают или уменьшают значение переменной, а в выражении возвращают значение переменной *$a* до изменения.

```
<?PHP
    $a=10;
    $b=$a++;
    echo   "a=$a,   b=$b";   //   Выводит   a=11,   b=10
?>
```

Результаты выполнения скрипта представлены на рис. 3.2.

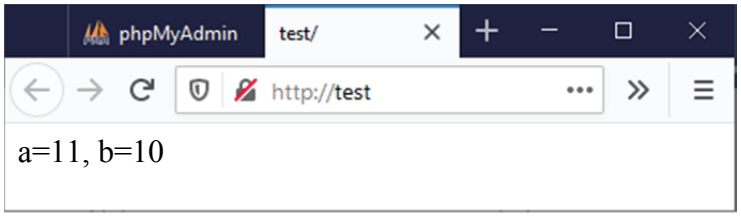

Рис. 3.2. Результаты применения постфиксного инкремента

Как видите, сначала переменной *$b* присвоилось значение переменной *$a*, а уж затем последняя была инкрементирована. Впрочем, выражение, значение которого присваивается переменной *$b*, может быть и сложнее – в любом случае, инкремент *$a* произойдет только после его вычисления.

Существуют также операторы инкремента и декремента, которые указываются до, а не после имени переменной (префиксные операторы инкремента и декремента). Соответственно, и возвращают они значение переменной уже после изменения.

```
<?PHP
    $a=10;
    $b=--$a;
    echo "a=$a, b=$b"; // Выводит a=9, b=9
?>
```

Результаты выполнения скрипта представлены на рис. 3.3.

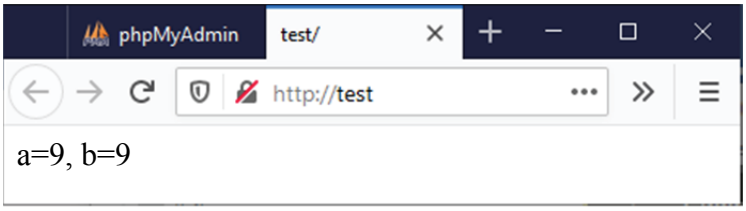

Рис. 3.3. Результаты применения префиксного декремента

Операции инкремента и декремента на практике применяются очень часто. Например, они встречаются практически в любом цикле *for* (будет рассмотрен в гл. 7).

Приведем другие примеры операций инкремента и декремента.

```
<?PHP
    echo "<b>Постфиксный инкремент</b><br>";
    $a = 5;
    echo "Должно быть 5: " . $a++ . "<br>";
    echo "Должно быть 6: " . $a . "<br>";
    echo "<b>Префиксный инкремент</b><br>";
    $a = 5;
    echo "Должно быть 6: " . ++$a . "<br>";
    echo "Должно быть 6: " . $a . "<br>";
    echo "<b>Постфиксный декремент</b><br>";
    $a = 5;
    echo "Должно быть 5: " . $a-- . "<br>";
    echo "Должно быть 4: " . $a . "<br>";
    echo "<b>Префиксный декремент</b><br>";
    $a = 5;
    echo "Должно быть 4: " . --$a . "<br>";
    echo "Должно быть 4: " . $a . "<br>";
?>
```

Результаты выполнения скрипта представлены на рис. 3.4.

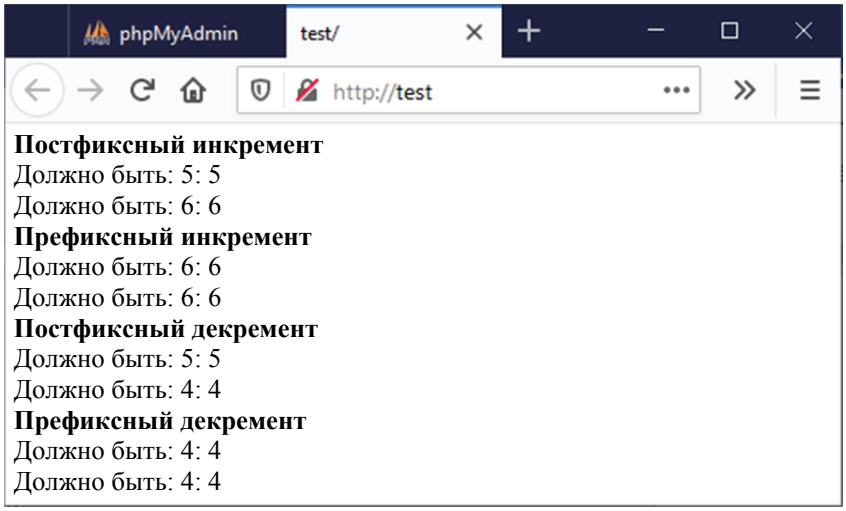

Рис. 3.4. Результаты использования инкремента и декремента

Операции инкремента можно применять и для символьных переменных. PHP следует соглашениям Perl (в отличие от C) касательно выполнения арифметических операций с символьными пе-

ременными. Так, например, в языке Perl *'Z'*+1 будет вычислено как *'AA'*, в то время как в языке C *'Z'*+1 будет вычислено как *'['* (ord('Z') == 90, ord('[') == 91). Это означает, что в PHP реализован не механизм увеличения кода ASCII строкового символа (коды ASCII представлены в прил. Б), а более логичная операция с точки зрения работы с символами – сдвиг по алфавиту (фактически берется следующая буква в алфавите). При этом различается регистр символов, т. е. «A» будет переходить в «B», а «a» будет при применении операции инкремента переходить в «b».

Пусть необходимо применить ее к строке abc.

```
<?PHP
    $a='abc';
    $b=$a++;
    echo "a=$a, b=$b"; // Выводит a=abd, b=abc
?>
```

Результаты выполнения скрипта представлены на рис. 3.5.

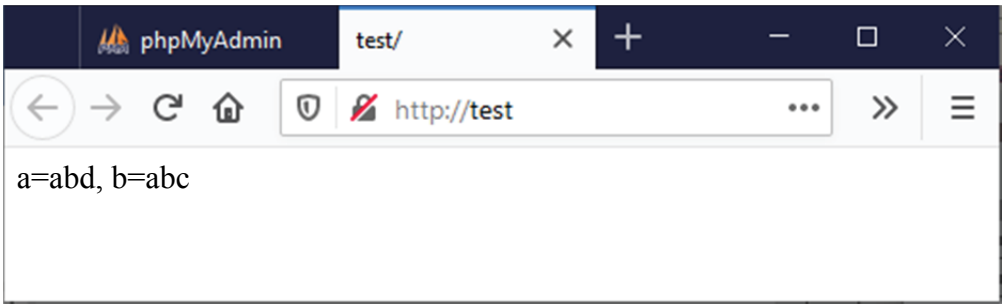

Рис. 3.5. Результат применения операции
инкрементирования к строковой переменной

Возможен вариант и префиксного инкремента.

```
<?PHP
    $a='a';
    echo 'a='.++$a,"<br>"; // Выводит a=b
    echo 'a='.$a++,"<br>"; // Выводит a=b, а затем а
увеличивает на 1
    echo 'a='.$a;
?>
```

Результаты выполнения скрипта представлены на рис. 3.6.

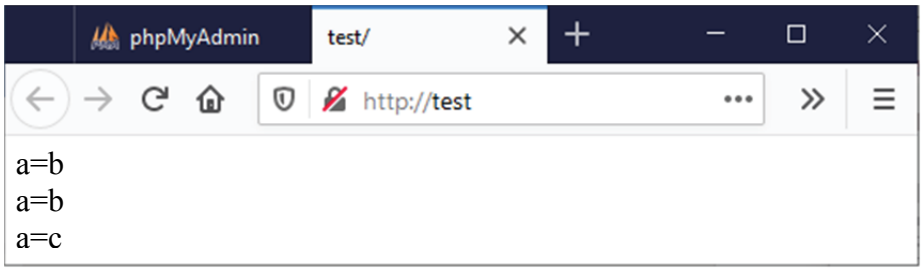

Рис. 3.6. Результат применения операции
перфиксного инкремента к строковой переменной

Отметим, что символьный тип подлежит только операции инкрементирования. Кроме того, поддерживается только алфавит ASCII и цифры (a–z, A–Z и 0–9). А вот булевый тип не подлежит ни инкрементированию, ни декрементированию.

3.3. Операции присвоения

Базовый оператор присвоения обозначается как «=» и означает, что левый операнд получает значение правого выражения, т. е. устанавливается результирующим значением.

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

```
<?PHP
    $b = 4;
    $a = $b  + 5; // результат: $a установлена значени-
ем 9, переменной $b присвоено 4.
    echo "a= $a , b= $b";
?>
```

Результаты выполнения скрипта представлены на рис. 3.7.

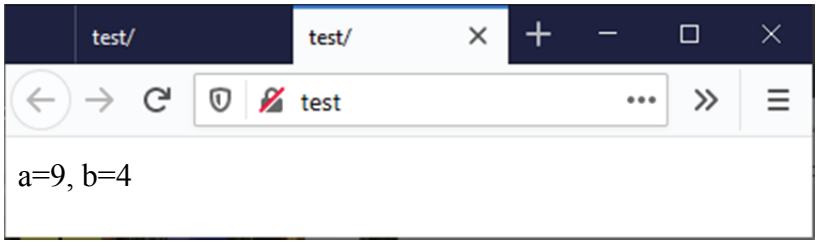

Рис. 3.7. Результат применения операций присваивания

Отметим, что в одной строке кода может быть несколько операций присвоения. В таком случае предыдущий пример будет выглядеть следующим образом:

```
<?PHP
    $a = ($b=4)  + 5; // результат: $a установлена зна
чением 9, переменной $b присвоено 4.
    echo "a= $a , b= $b";
?>
```

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

```
<?PHP
    $a = 3;
    $a += 5; // устанавливает $a значением 8, аналогич-
но записи: $a = $a + 5;
    echo "a=$a";
echo"<br>";
    $a *= 5; // устанавливает $a значением 40, анало-
гично записи: $a = $a * 5;
    echo "a=$a";
echo"<br>";
    $a /= 4; // устанавливает $a значением 10, анало-
гично записи: $a = $a / 4;
    echo "a=$a";
    echo"<br>";
?>
```

Результаты выполнения скрипта представлены на рис. 3.8.

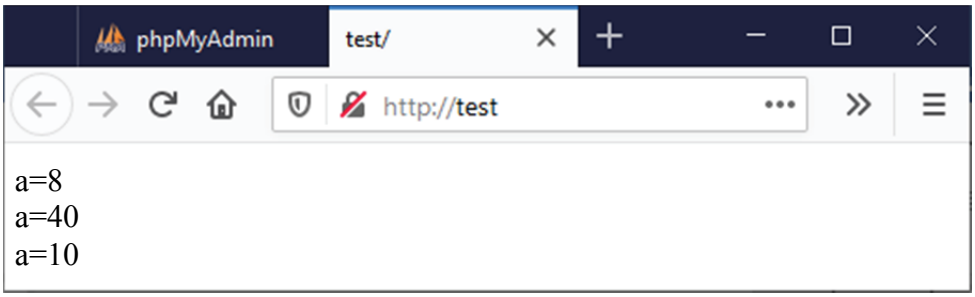

Рис. 3.8. Результат применения комбинированных операций

Обратите внимание, что присвоение копирует оригинальную переменную в новую (присвоение по значению), таким образом все последующие изменения одной из переменных на другой никак не отражаются, т. е. связи между переменными нет.

Начиная с PHP 7 добавился новый оператор «??» (null coalescing), который позволяет присваивать значение по умолчанию. Приведем простой пример.

```
<?PHP
     // Пример использования оператора
     $action = $_POST['action'] ?? 'default';
     echo $action;
     echo '<hr>';
     // Пример выше аналогичен следующему коду
     if (isset($_POST['action'])) {
         $action = $_POST['action'];
     } else {
         $action = 'default';
     }
     echo $action;
?>
```

Результаты выполнения скрипта представлены на рис. 3.9.

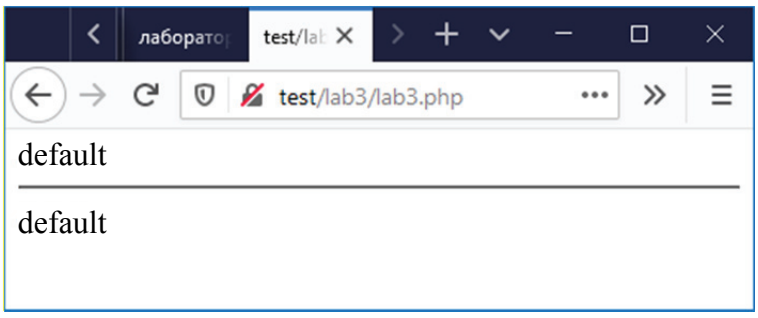

Рис. 3.9. Результат применения оператора «??»

Мы видим, что если не задано значение переменной, ей присваивается значение по умолчанию (default).

3.4. Операции со строками (конкатенация)

В PHP есть два оператора для работы со строками. Первый – оператор конкатенации « . », который возвращает объединение ле-

вого и правого аргумента. Второй – оператор присвоения с конкатенацией, который присоединяет правый аргумент к левому. Приведем несколько примеров.

```
<?PHP

    $a = "Hello ";
    $b = $a . "World!"; // $b содержит строку "Hello
World!"
    echo $b;
    echo"<br>";

    $a = "Hello ";
    $a .= "World!";     // $a содержит строку "Hello
World!"
    echo $a;

?>
```

Результаты выполнения скрипта представлены на рис. 3.10.

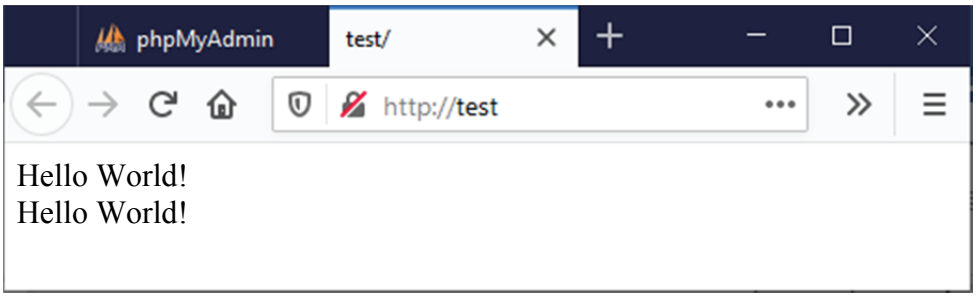

Рис. 3.10. Результат применения комбинированных операций со строковыми переменными

Как видно из последнего рисунка, результат обеих операций один и тот же.

3.5. Побитовые операции

Побитовые операции предназначены для работы (установки / снятия / проверки) групп бит в целой переменной. Биты целого числа – это не что иное, как отдельные разряды того же самого числа, записанного в двоичной системе счисления. Если переменная

не целая, то она в начале округляется, а уж затем к ней применяются перечисленные ниже операторы.

Для представления одного целого числа (тип integer) используются 32 бита:

0000 0000 0000 0000 0000 0000 0000 0000 – это ноль;
0000 0000 0000 0000 0000 0000 0000 0001 – это 1;
0000 0000 0000 0000 0000 0000 0000 0010 – это 2;
0000 0000 0000 0000 0000 0000 0000 0011 – это 3;
0000 0000 0000 0000 0000 0000 0000 0100 – это 4;
0000 0000 0000 0000 0000 0000 0000 0101 – это 5;
...
0000 0000 0000 0000 0000 0000 0000 1100 – это 12;
...
0000 0000 0000 0000 0000 0000 0000 1111 – это 15;
...

и т. д.

В PHP применяются следующие побитовые операторы (табл. 3.3).

Таблица 3.3

Таблица побитовых операций

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

Приведем примеры побитовых операций.

Пример 1. Побитовая операция 'И' 12 & 6.

$$12 \ \& \ 6 \rightarrow \frac{1100}{0110} \rightarrow 0100,$$

т. е. в десятичной форме 4.

Пример 2. Побитовая операция 'ИЛИ' 12 | 6.

$$12 \mid 6 \rightarrow \frac{1100}{0110} \rightarrow 1110,$$

т. е. в десятичной форме 14.

Пример 3. Побитовая операция исключающее 'ИЛИ' 12 ^ 6 .

$$12 \ \hat{} \ 6 \rightarrow \frac{1100}{0110} \rightarrow 1010,$$

т. е. в десятичной форме 10.

Пример 4. Операция сдвига влево (все числа в PHP кодируются 32 битами) 13<<2.

$$13 << 2 \rightarrow 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101 << 2 \rightarrow$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011\ 0100,$$

т. е. в десятичной форме 52.

$$13 >> 2 \rightarrow 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101 >> 2 \rightarrow$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011,$$

т. е. в десятичной форме 3.

Рассмотрим подобные примеры, написанные на языке PHP.

```
<?PHP
    $a = 12; //в двоичной системе 12 - это 1100
    $b=$a|6;
    echo "12|6 = $b"; // результат = 14
    echo"<br>";

    $b=$a&6;
    echo "12&6 = $b";  // результат = 4
    echo "<br>";

    $b=$a^6;
    echo "12^6 = $b"; // результат = 10
    echo"<br>";

    $b=$a<<2;
    echo "$a<<2 = $b"; // результат = 48
    echo"<br>";

    $b=++$a<<2;
    echo "$a<<2 = $b"; // результат = 52
    echo"<br>";
```

```
    $b=$a>>2;
    echo "$a>>2 = $b"; // результат = 3
    echo"<br>";

    $b=$a>>3;
    echo "$a>>2 = $b"; // результат = 1
    echo"<br>";
?>
```

Результаты выполнения скрипта представлены на рис. 3.11.

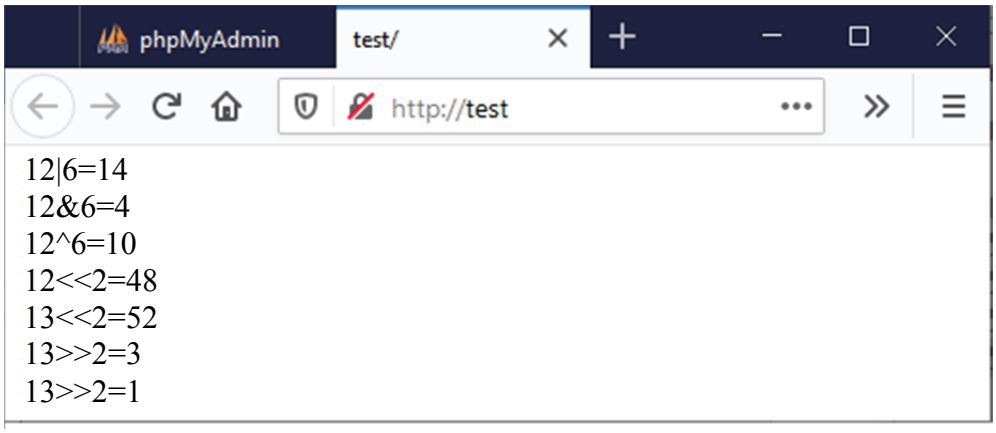

Рис. 3.11. Результат использования побитовых операций

Приведены значения, операция, выполняемая над значениями, и результат ее выполнения.

3.6. Операции сравнения

Операторы сравнения, как это видно из их названия, позволяют сравнивать между собой два значения.

Это в своем роде уникальные операции, потому что независимо от типов своих аргументов, они всегда возвращают одно из двух: *false* или *true*. Операции сравнения позволяют сравнивать два значения между собой и, если условие выполнено, возвращают *true*, а если нет – *false*.

В PHP разрешается сравнивать только скалярные переменные. Объекты в PHP сравнивать нельзя. Применяются следующие операторы сравнения (табл. 3.4).

Таблица 3.4

Таблица побитовых операций

Пример	Название	Результат
$*a* == $*b*	Равно	TRUE если $*a* равно $*b*
$*a* === $*b*	Тождественно равно	TRUE если $*a* равно $*b* и имеет тот же тип
$*a* != $*b*	Не равно	TRUE если $*a* не равно $*b*
$*a* <> $*b*	Не равно	TRUE если $*a* не равно $*b*
$*a* !== $*b*	Тождественно не равно	TRUE если $*a* не равно $*b* или если они разных типов (добавлено в PHP 4 и старше)
$*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*
$*a* <=> $*b*	spaceship	Число типа integer меньше, больше или равное нулю при *$a* соответственно меньше, больше или равно *$b* (доступно с PHP 7)

Приведем несколько примеров.

```
<?PHP
    $a = 10.1+0.9; //переменная типа float
    $b=11; //переменная типа integer
    echo 'Результат условия $a==$b : ';
    var_dump(($a==$b));
    echo "<br>";
    echo 'Результат условия $a===$b : ';
    var_dump(($a===$b));
    echo "<br>";
    echo 'Результат условия $a!==$b : ';
    var_dump(($a!==$b));
?>
```

Результаты выполнения скрипта представлены на рис. 3.12.

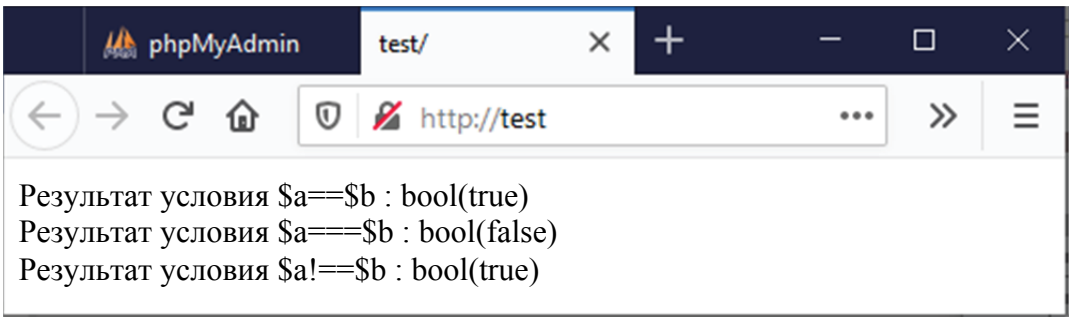

Рис. 3.12. Результат выполнения логических операций

Интересным является и тот факт, что при сравнении бесконечной дроби 1/3 с числом 0.33333333333333 (переменная типа float (15 знаков после запятой)), результат будет *false*, а если записать данное число с 16 знаками после запятой, то результат становится *true*. Связано это с правилами представления (хранения) в памяти в двоичном виде переменных типа *float*.

```
<?PHP

    $a = 1/3; //переменная типа float
    $b=0.333333333333333; //переменная типа float (14
знаков после запятой)
    echo 'Результат условия $a==$b (15 знаков после за-
пятой): ';
    var_dump(($a==$b));
    echo "<br>";
    $b=0.3333333333333333; //переменная типа float (16
знаков после запятой)
    echo 'Результат условия $a==$b (16 знаков после за-
пятой): ';
    var_dump(($a==$b));
?>
```

Результаты выполнения скрипта представлены на рис. 3.13.

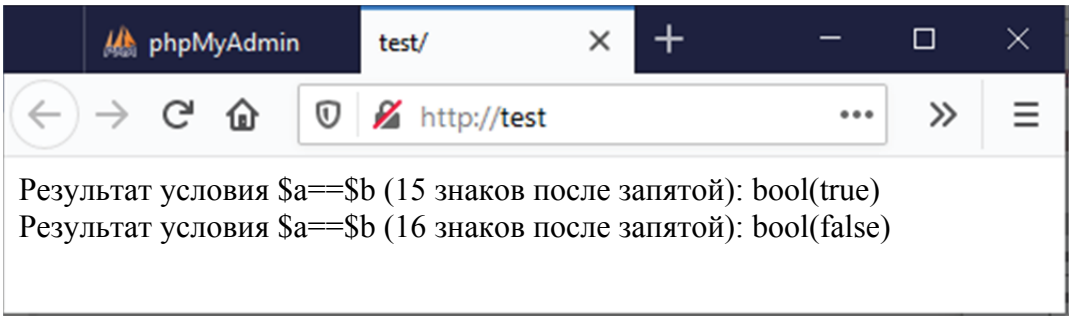

Рис. 3.13. Пример, показывающий точность операций с плавающей точкой

Далее рассмотрим оператор тождественного сравнения – тройной знак равенства ===, или оператор проверки на эквивалентность. Необходимо отметить, PHP довольно терпимо относится к тому, что строки неявно преобразуются в числа, и наоборот. Например, следующий код выведет, что значения переменных равны.

```
<?PHP
    $a=10;
    $b="10";
    var_dump($a==$b); // true
?>
```

Результаты выполнения скрипта представлены на рис. 3.14.

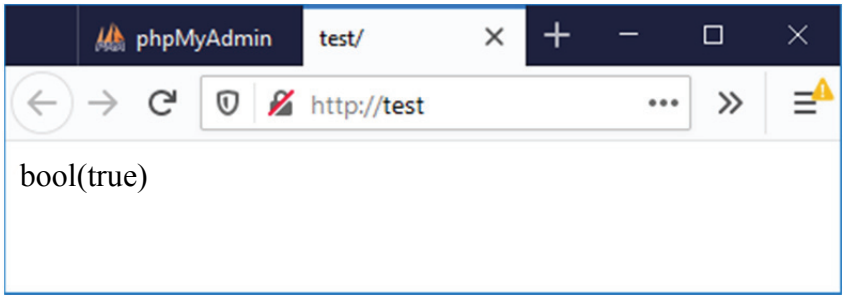

Рис. 3.14. Пример сравнения переменных разных типов

И это несмотря на то, что переменная *$a* представляет собой число, а *$b* – строку. Теперь рассмотрим несколько другой пример:

```
<?PHP
    $a=0;
    $b="";
    var_dump($a==$b); // true
?>
```

Результаты выполнения скрипта представлены на рис. 3.15.

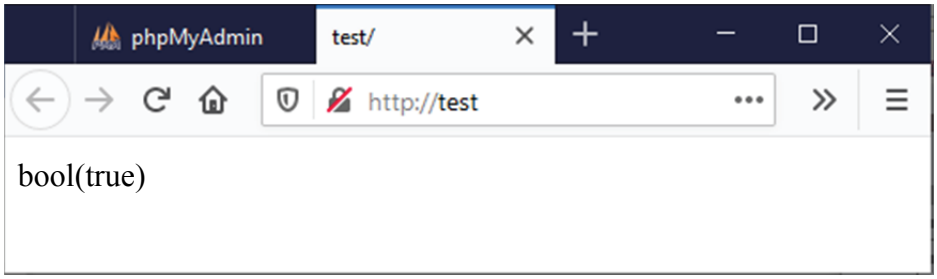

Рис. 3.15. Пример сравнения переменных integer и string

Хотя $*a* и $*b* явно неравны даже в обычном понимании этого слова, результат выполнения скрипта показывает, что они совпадают. Дело в том, что если один из операндов логического оператора может трактоваться как число, то оба операнда трактуются как числа. При этом пустая строка превращается в 0, который и

сравнивается с нулем. В итоге при сравнении результата получаем *true*. Проблему решает оператор эквивалентности === (тройное равенство). Он не только сравнивает два выражения, но также их типы. Перепишем пример с использованием этого оператора.

```
<?PHP
    $a=0;
    $b="";
    var_dump($a===$b); // false
?>
```

Результаты выполнения скрипта представлены на рис. 3.16.

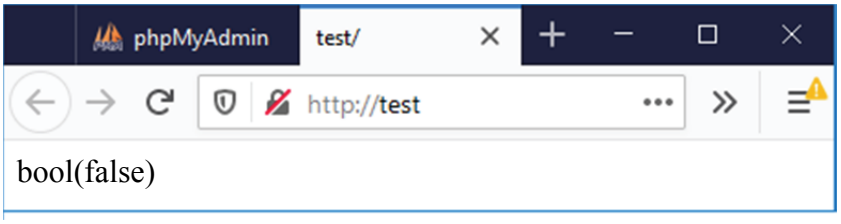

Рис. 3.16. Пример сравнения переменных integer и string с использованием операции «===»

Вот теперь результат сравнения равен *false*. Но возможности оператора эквивалентности идут далеко за пределы сравнения простых переменных. С его помощью можно сравнивать также массивы, объекты и т. д. Иногда это очень удобно.

Что касается сравнения массивов, то необходимо обратить внимание на следующие моменты. Основные операторы, применяющиеся для данных целей, представлены в табл. 3.4.

Таблица 3.4

Операторы сравнения, работающие с массивами

Пример	Название	Результат
$*a* == $*b*	Равно	**TRUE** в случае, если *$a* и *$b* содержат одни и те же пары ключ / значение
$*a* === $*b*	Тождественно равно	**TRUE** в случае, если *$a* и *$b* содержат одни и те же пары ключ / значение в том же самом порядке и того же типа
$*a* != $*b*	Не равно	**TRUE**, если массив *$a* не равен массиву *$b*
$*a* <> $*b*	Не равно	**TRUE**, если массив *$a* не равен массиву *$b*
$*a* !== $*b*	Тождественно не равно	**TRUE**, если массив *$a* не равен тождественно массиву *$b*

Вот пример сравнения двух массивов с одинаковыми значениями элементов (но есть различия в парах ключ / значение).

```
<?php
    $A = array("apple", "banana", "pear");
    $B = array(1=>"banana", 0=>"apple", 2=>"pear");
    echo 'массив A'.'<br>';
        foreach ($A as $key => $value) {
        echo $key.' => '.$value.'<br>';
    }
    echo '<br><br>';
     echo 'массив B'.'<br>';
        foreach ($B as $key => $value){
        echo $key.' => '.$value.'<br>';
    }
    echo "<hr>";
    echo 'Результат условия $a==$b : ';
    var_dump(($A == $B)); // bool(true)
    echo '<br>'.'Результат условия $a===$b : ';
    var_dump(($A === $B)); // bool(false)
?>
```

Результаты работы скрипта представлены на рис. 3.17.

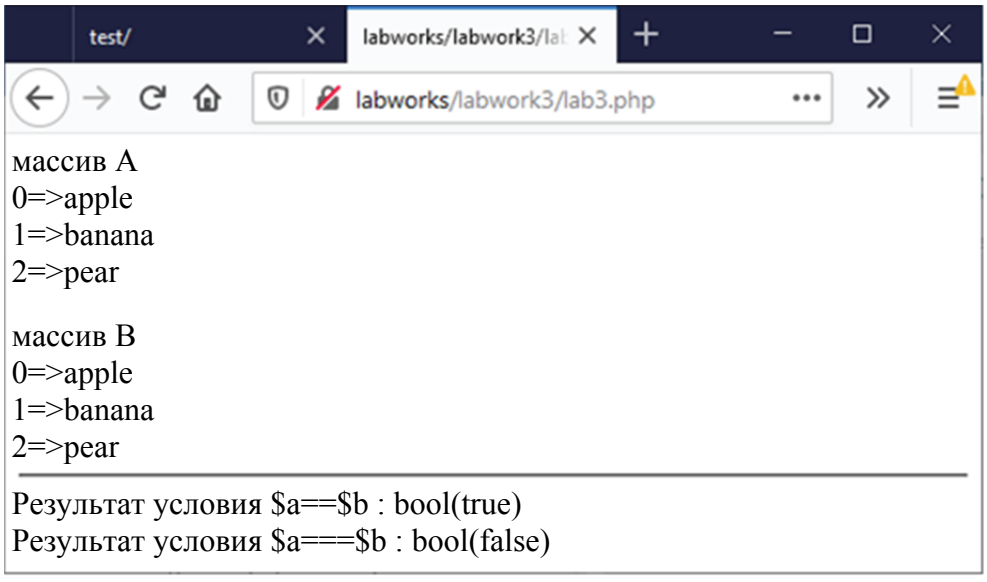

Рис. 3.17. Результаты сравнения массивов

Результаты работы скрипта при сравнении через «===» на первый взгляд кажутся странными, так как и сами значения, пары

ключ / значение совпадают. Однако необходимо обратить внимание и на последовательность хранения пар ключ / значение в массиве. Фактически при проверке эквивалентности пара *0 => "apple"* массива *A* сравнивается с *1 => "banana"* массива *B*. Поэтому при проверке эквивалентности результат *false*.

Аналогичные результаты получим при сравнении казалось бы одинаковых массивов $A = array("1", "2", "3") и $B = array(1, 2, 3), но отличающихся типами данных.

Также необходимо отметить, что для оператора «===» существует и его антипод – оператор «!===» .

3.7. Логические операции

Логические операторы предназначены исключительно для работы с логическими выражениями и также возвращают *false* или *true*. В табл. 3.5 представлены логические операции, используемые на PHP.

Отметим, что "| |" имеет больший приоритет, чем "OR", а "&&" – чем "AND".

Таблица 3.5

Таблица логических операций

Пример	Название	Результат
$*a* and $*b*	Логическое 'И'	TRUE если и $*a*, и $*b* TRUE
$*a* or $*b*	Логическое 'ИЛИ'	TRUE если или $*a*, или $*b* TRUE
$*a* xor $*b*	Исключающее 'ИЛИ'	TRUE если $*a*, или $*b* TRUE, но не оба
! $*a*	Отрицание	TRUE если $*a* не TRUE (инвертирует логическое значение операнда)
$*a* && $*b*	Логическое 'И'	TRUE если и $*a*, и $*b* TRUE
$*a* || $*b*	Логическое 'ИЛИ'	TRUE если или $*a*, или $*b* TRUE

Далее рассмотрим несколько примеров.

```
<?PHP
    $a = 10;
    $b=-5;
    echo 'Результат условия ($a>0 and $b>0) : ';
    var_dump(($a>0 and $b>0));
    echo "<br>";
    echo 'Результат условия ($a>0 || $b>0) : ';
```

```
    var_dump(($a>0 || $b>0));
    echo "<br>";
    $b=5;
    echo 'Результат условия ($a>0 xor $b>0) : ';
    var_dump(($a>0 xor $b>0));
    echo "<br>";
?>
```

Результаты выполнения скрипта представлены на рис. 3.18.

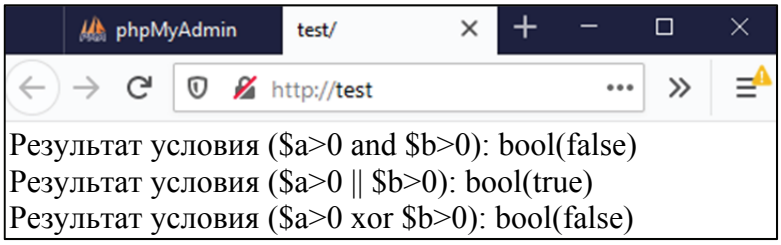

Рис. 3.18. Результаты применения логических операций

Следует заметить, что вычисление логических выражений, содержащих представленные в таблице операторы, идет всегда слева направо, при этом если результат уже очевиден (например, *false&&что-то* всегда дает *false*), то вычисления обрываются, даже если в выражении присутствуют вызовы функций. Рассмотрим следующий пример:

```
<?PHP
    $a = 10;
    $b=-5;
    echo 'Результат условия ($a<0 and ++$b<0) : ';
    var_dump(($a<0 and ++$b<0));
    echo "<br>";
    var_dump($b);
    echo "<br>";
    echo 'Результат условия ($a>0 and ++$b<0) : ';
    var_dump(($a>0 and ++$b<0));
    echo "<br>";
    var_dump($b);
?>
```

В связи с тем, что в составном условии ($a<0 and ++$b<0) первая часть не выполнилась, а это означает, что при объединении через *and* общий результат уже обязательно будет *false*, операция префиксного инкремента переменной *b* не выполнялась (рис. 3.19).

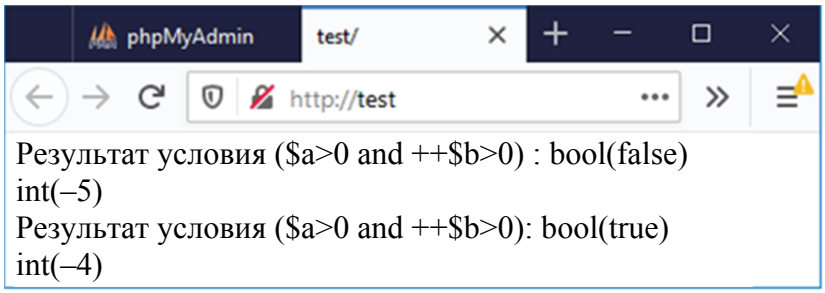

Рис. 3.19. Результаты применения нескольких логических операций в одном условии

Будьте осторожны с логическими операциями – не забывайте про удваивание символа. Обратите внимание, что, например, | и || – два совершенно разных оператора, один из которых может потенциально возвращать любое число, а второй – только **false** и **true**.

3.8. Приоритеты операторов

Для корректного понимания результатов математических и логических операций важно знать приоритеты каждой их них (табл. 3.6). Операторы с более высоким уровнем приоритета выполняются в первую очередь.

Таблица 3.6

Приоритеты операций

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

Рассмотрим следующий пример:

```
<?PHP
    $a=11;
    $b=$a<<2+5;
    echo 'Значение b='.$b;
?>
```

Результат работы скрипта представлен на рис. 3.20.

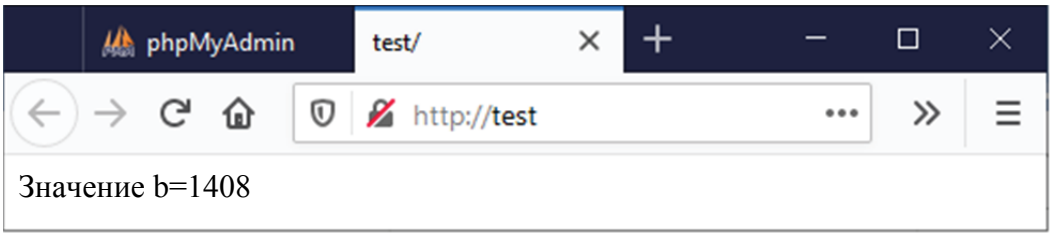

Рис. 3.20. Результат применения нескольких операций
с разными приоритетами в одном выражении

В соответствии с приоритетами операций, представленными в табл. 3.6, сначала выполнится сложение 2+5, а уже затем к значению переменной $*a*=11 будет применен сдвиг влево на 7 бит. В результате получим 1408.

Практическая часть

Пример 1

Рассмотрим пример решения системы линейных алгебраических уравнений методом Крамера. Код сохраним в файле lab3.php. В примере PHP-скрипт встроен в HTML-старницу.

```
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <h2 align="center">Решение системы линейных уравне-
ний</h2>
    <p align="center">
```

```
      2<i>x</i>+3<i>y</i>=8<br />
      3<i>x</i>+2<i>y</i>=7
    </p>
    <?PHP
      $a11=2; $a12=3; $b1=8;
      $a21=3; $a22=2; $b2=7;
      $delta=$a11*$a22-$a12*$a21;
      $deltaX=$b1*$a22-$a12*$b2;
      $deltaY=$a11*$b2-$b1*$a21;
      $x=$deltaX/$delta;
      $y=$deltaY/$delta;
      echo   "<p   align='center'>Определеители<br   />
&Delta; = $delta<br />";
      echo "&Delta;<sub><i>x</i></sub> = $deltaX 
 ";
      echo   "&Delta;<sub><i>y</i></sub>   =   $deltaY<br
/>";
      echo "Ответы<br /><i>x</i>=$x  <i>y</i>
=$y</p>";
    ?>
  </body>
</html>
```

Для запуска скрипта на выполнение необходимо в адресной строке браузера указать название файла скрипта lab3.php. Если в качестве сервера используется встроенный в PHP, то в строке браузера набираем localhost:4000/lab3.php. Результаты выполнения скрипта представлены на рис. 3.21.

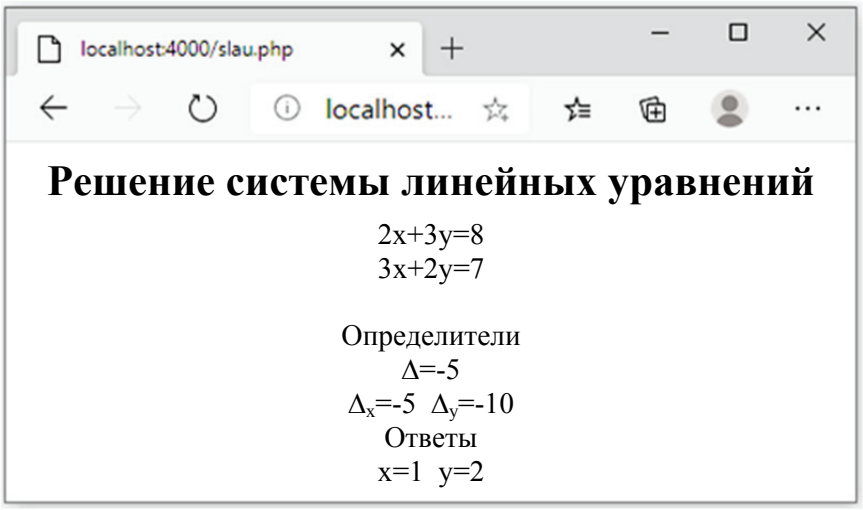

Рис. 3.21. Результат решения СЛАУ

Пример 2

Рассмотрим пример оперций со строковыми переменными, а именно инкремента.

```
<?PHP
    echo 'ASCII код символа Z= '.ord("Z"),"<br>";
    echo 'ASCII код символа AA= '.ord("AA"),"<br>";
    echo 'ASCII код символа [= '.ord("["),"<br>";
    echo"<br>";

    $i = 'W';
    echo "Начальное значение строки = $i";
    echo"<br>";
    for($n=0; $n<6; $n++)
    {
    echo ++$i;
    echo"<br>";
    }
    echo"<br>";

    $i = 'AAZ';
    echo "Начальное значение строки = $i";
    echo"<br>";
    for($n=0; $n<6; $n++)
    {
    echo ++$i;
    echo"<br>";
    }
    echo"<br>";
    $i = 'ADz';

    echo "Начальное значение строки = $i";
    echo"<br>";
    echo "Результат инкремента ADz = ".++$i;
?>
```

Результат работы скрипта представлен на рис. 3.22. Отметим, что PHP корректно работает, даже если в одной строке присутствуют символы как в верхнем, так и нижнем регистре.

Как видно из рисунка, при операции инкремента происходит сдвиг на следующую букву алфавита в младшем разряде строки. Если же такое невозможно, т. е. при использовании латинского алфавита в младшем разряде уже стоит «Z», то в данном разряде пишется младшая буква алфавита, а сдвиг на следующую букву

алфавита выполняется в предшествующем разряде. Поэтому при инкременте «AAZ» получили «ABA». В русском или любом другом языке преобразования происходят точно по таким же правилам.

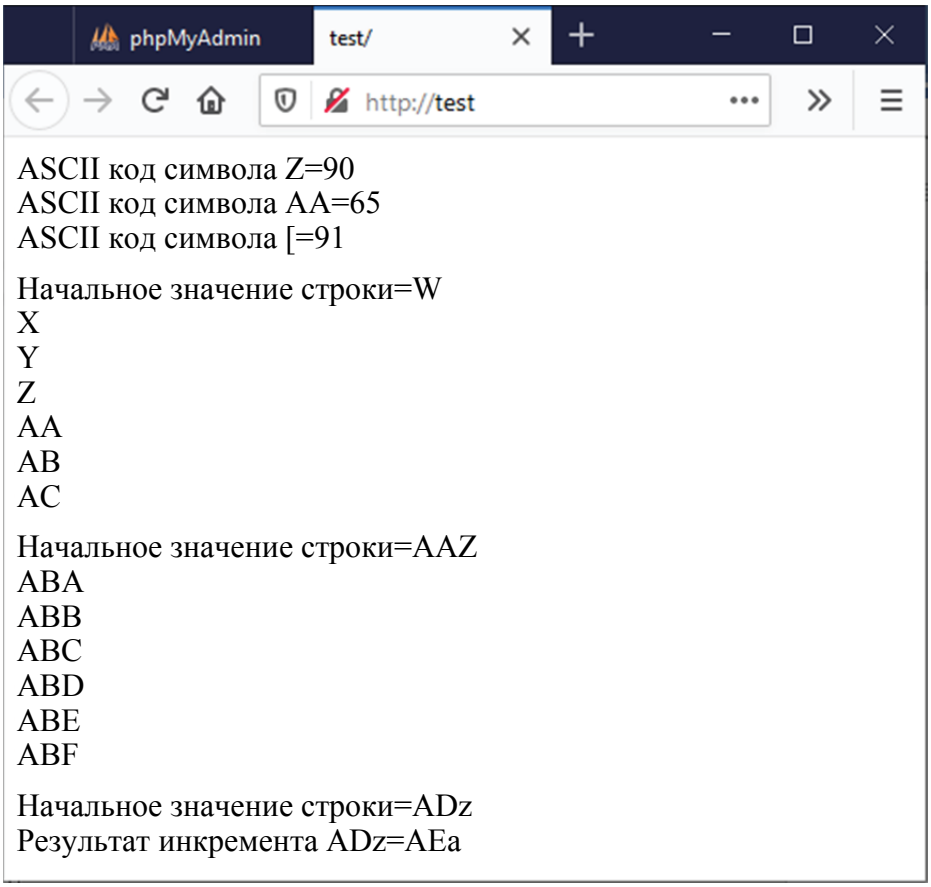

Рис. 3.22. Результат применения операции инкремента к строковым переменным

Самостоятельно проанализируйте, что произойдет при достижении самого последнего символа алфавита.

Лабораторная работа № 3

Задания на лабораторную работу будут носить похожий на рассмотренные примеры характер и выдаваться индивидульно преподавателем. Напомним, что выполнение и демострация всех лабораторных работ предполагается в пределах структуры, представленной в лабораторной работе № 1.

ГЛАВА 4

МАТЕМАТИЧЕСКИЕ ОПЕРАЦИИ

Большинство математических действий в языке PHP осуществляются в форме встроенных функций, а не в форме операций. Кроме операций сравнения (были рассмотрены в гл. 3), язык PHP предлагает пять операций выполнения простых арифметических действий, а также некоторые сокращенные операции, позволяющие составлять более краткие выражения инкремента и декремента, а также присваивания.

Ранее (в гл. 3) были рассмотрены примеры пяти стандартных математических операций (сложение, вычитание, умножение и деление, а также остаток от деления) и операций инкремента / декремента. Повторно мы их рассматривать не будем, а перейдем к другим математическим функциям.

4.1. Простые математические функции

Простые математические функции позволяют выполнять такие задачи, как преобразование из одного числового типа в другой, поиск минимального или максимального числа во множестве чисел и т. д. В табл. 4.1 представлены простые математические функции.

Таблица 4.1

Простые математические функции

Функция	Описание
floor()	Принимает единственный фактический параметр (как правило, число с плавающей точкой двойной точности) и возвращает наибольшее целое число, которое меньше или равно этому фактическому параметру (округление в меньшую сторону)
ceil()	Имя этой функции представляет собой сокращение от слова ceiling (потолок). Функция принимает единственный фактический параметр (как правило, число с плавающей точкой) и возвращает наименьшее целое число, которое больше или равно этому фактическому параметру (округление в большую сторону)
round()	Принимает единственный фактический параметр (как правило, число с плавающей точкой двойной точности) и возвращает ближайшее целое число

Окончание табл. 4.1

Функция	Описание
abs()	Модуль числа. Если единственный числовой фактический параметр имеет отрицательное значение, то функция возвращает соответствующее положительное число; если фактический параметр является положительным, то функция возвращает сам фактический параметр
min()	Принимает любое количество числовых фактических параметров (но не менее одного) и возвращает наименьшее из всех значений фактических параметров
max()	Принимает любое количество числовых фактических параметров (но не менее одного) и возвращает наибольшее из всех значений фактических параметров
sqrt()	Вычисляет квадратный корень

Рассмотрим примеры применения математических функций.

```
<?PHP
    var_dump (abs(-3));
    var_dump (round(2.7));
    var_dump (ceil(2.3));
    var_dump (floor(3.9));
    $result = min(2, abs(-3), -3,ceil(2.3) );
    echo 'минимальное значение= '.$result;
    $result = max(2, abs(-3), -3,ceil(2.3) );
    echo "<br>";
    echo 'максимальное значение = '.$result;
?>
```

Результат работы скрипта представлен на рис. 4.1.

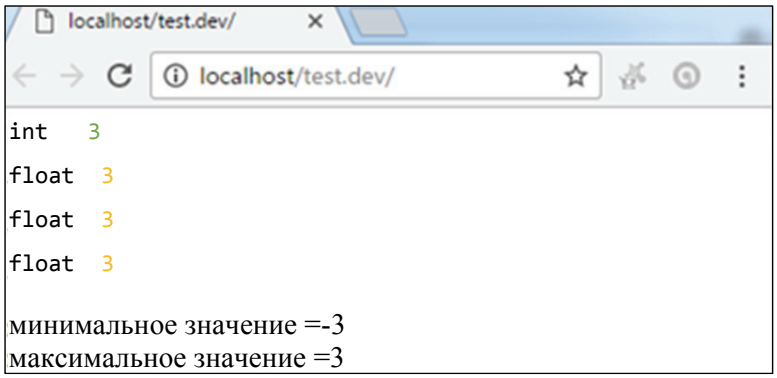

Рис. 4.1. Пример использования математических функций

Самостоятельно проанализируйте полученные результаты.

4.2. Выработка случайных чисел

В языке PHP применяются два генератора случайных чисел (вызываемых соответственно с помощью функций rand() и mt_rand()). С каждым из этих генераторов связаны по три функции одинакового назначения: функция задания начального значения (srand() и mt_srand()), сама функция получения случайного числа и функция, осуществляющая выборку наибольшего целого числа, которое может быть возвращено генератором ((getrandmax() и mt_getrandmax())). Функции getrandmax() и mt_getrandmax() возвращают значение наибольшего числа, которое может быть возвращено функцией rand() или mt_rand(). Значение зависит от вида и разрядности операционной системы.

Выбор конкретной функции выработки псевдослучайных чисел, которая используется в функции rand(), может зависеть от того, с какими именно библиотеками был откомпилирован интерпретатор PHP. В отличие от этого, в генераторе mt_rand() всегда используется одна и та же функция выработки псевдослучайных чисел (mt – сокращение от Mersenne Twister), причем автор оперативной документации к функции mt_rand() утверждает, что эта функция к тому же является более быстродействующей и «более случайной» (с точки зрения криптографии), чем rand().

```
<?PHP

    for ($i=0; $i<=10; $i++)
    {
        echo rand();
        echo "<br>";
    }

?>
```

Результаты работы скрипта представлены на рис. 4.2. Отметим, что на рисунке приведены два скриншота, на которых получены разные значения.

Рассмотрим аналогичный пример, но с применением функции mt_rand().

```
<?PHP
    for ($i=0; $i<=10; $i++)
    {
        echo mt_rand();
        echo "<br>";
    }
?>
```

Результат работы скрипта представлен на рис. 4.3.

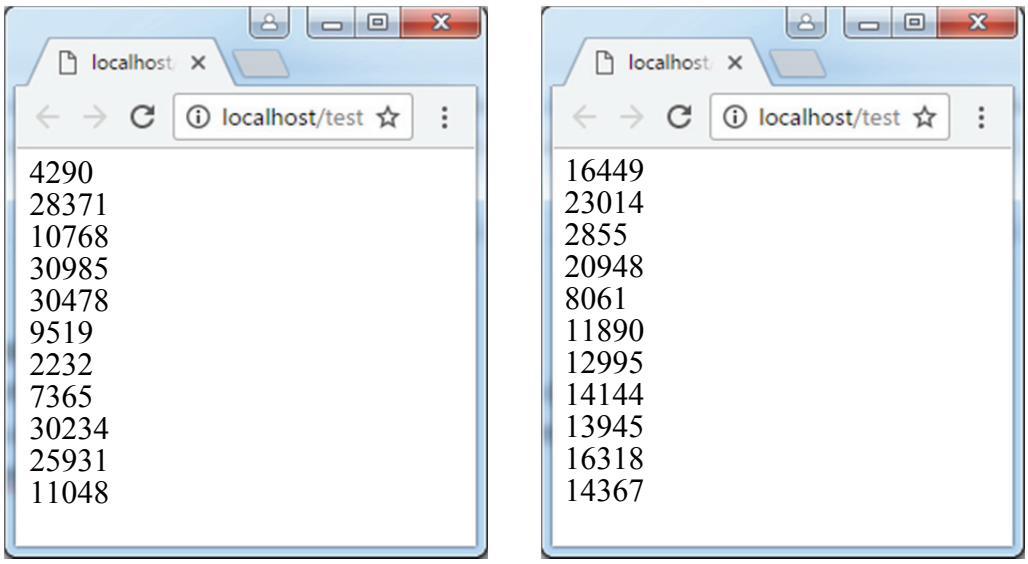

Рис. 4.2. Пример генерации случайных чисел

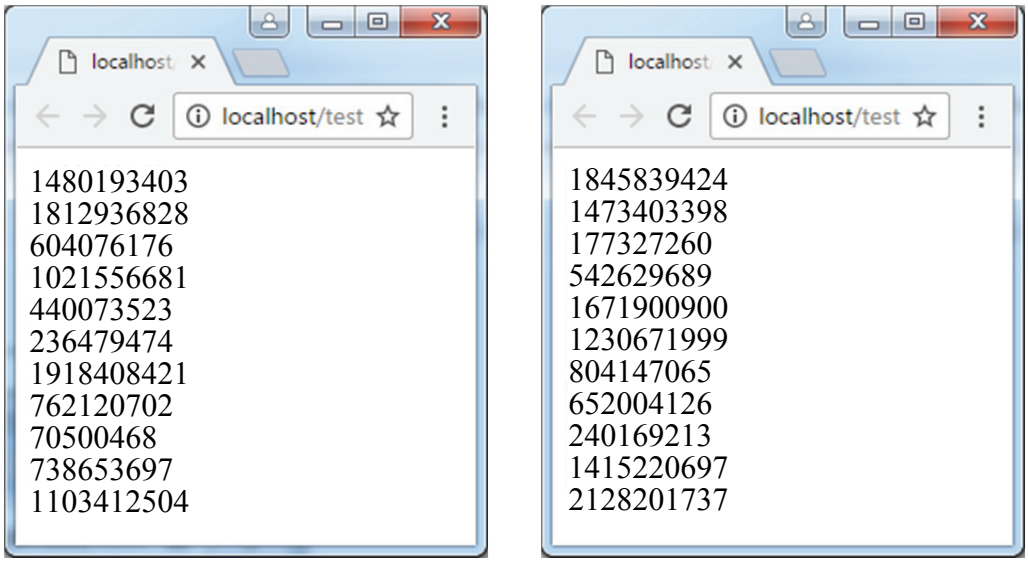

Рис. 4.3. Генерация случайных чисел
с использованием функции mt_rand()

При использовании некоторых версий PHP для отдельных платформ создается впечатление, что функции rand() и mt_rand() вырабатывают на первый взгляд вполне приемлемые случайные числа, даже без предварительного задания начального значения. Но такому впечатлению не следует доверять. Во-первых, программы, в которых используются функции выработки случайных чисел без задания начального значения, невозможно легко переносить на другие платформы, а, во-вторых, надежная работа указанных функций без задания начального значения не гарантируется. Типичный способ задания начального значения для любого из генераторов случайных чисел PHP (с использованием функции mt_srand() или srand()) заключается в следующем. В этом операторе задается начальное значение генератора, равное количеству микросекунд, истекших к данному времени с момента отсчета последней целой секунды. (Операция «Приведение типа» к типу float необходима, поскольку функция microtime() возвращает строку, которая рассматривается как целое число в операции умножения, но не в операции передачи параметров в функцию.)

```
<?PHP
    mt_srand((float)(microtime()*1000000));
    for ($i=0; $i<=10; $i++){
        echo mt_rand();
        echo "<br>";
    }
?>
```

Результат работы скрипта представлен на рис. 4.4.

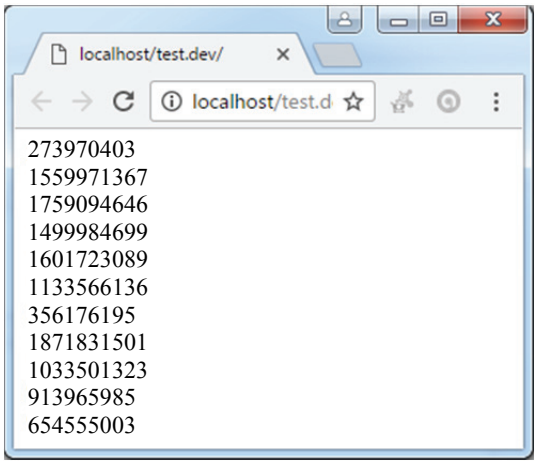

Рис. 4.4. Пример генерации случайных чисел
с предварительно заданным начальным значением

Отметим, что достаточно просто поместить данный оператор на каждую страницу PHP всего лишь один раз перед использованием соответствующей функции mt_rand() или rand(), и этот оператор будет гарантировать, что отправная точка изменится и поэтому каждый раз будут вырабатываться различные случайные последовательности.

Очевидно, что указанные функции выработки псевдослучайных чисел возвращают только целые числа, но случайное целое число из заданного диапазона можно легко преобразовать в соответствующее число с плавающей точкой (скажем, в число из диапазона от 0.0 до 1.0 включительно) с помощью выражения наподобие rand() / getrandmax(). После этого указанный диапазон можно масштабировать и сдвигать по мере необходимости. Ниже показан пример генерации целых случайных чисел от 100.0 до 120.0.

```
<?PHP
    mt_srand((float)(microtime()*1000000));
    for ($i=0; $i<=10; $i++){
        $random = 100.0 + 100.0 * mt_rand() /
mt_getrandmax();
        echo (int)$random;
        echo "<br>";
    }
?>
```

Результат работы скрипта представлен на рис. 4.5.

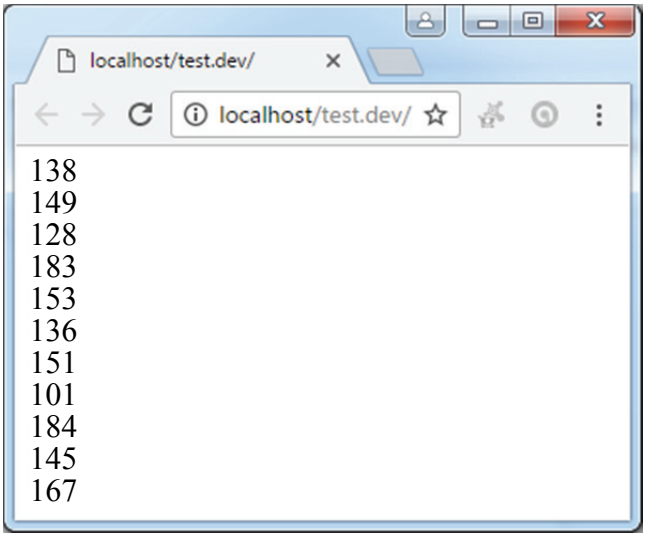

Рис. 4.5. Пример генерации случайных чисел в диапазоне от 100 до 120

Попробуйте обновить страницу с этим кодом несколько раз, чтобы убедиться в генерации случайных чисел.

4.3. Математические константы

Большинство математических констант, используемых в PHP, относятся к числу π (или к кратным ему значениям), числу *e* (или к кратным ему значениям), а также к квадратным корням; кроме того, некоторые константы могут относиться к другим типам (табл. 4.2).

Таблица 4.2

Математические константы PHP

Константа	Описание
M_PI	π
M_PI_2	π / 2
M_PI_4	π / 4
M_1_PI	1 / π
M_2_PI	2 / π
M_2_SQRTPI	2 / sqrt(π)
M_E	*e*
M_SQRT2	sqrt(2)
M_SQRT1_2	1 / sqrt(2)
M_LOG2E	$\log_2(e)$
M_LOG10E	lg(*e*)
M_LN2	$\log_e(2)$
M_LN10	$\log_e(10)$

Примеры использования математических констант будут представлены дальше.

4.4. Проверка формата чисел

В языке PHP предусмотрен ряд функций, позволяющих выполнять проверку правильности представления чисел. Несмотря на то, что в языке PHP отсутствует строгий контроль типов, рекомендуется

в случае необходимости применять некоторые из этих проверок в коде, чтобы иметь возможность прогнозировать характеристики полученных результатов, а также выбирать наилучший способ их обработки.

Первая и наиболее простая проверка заключается в использовании функции **is_numeric()**. Как и при осуществлении большинства других таких проверок, функция **is_numeric** возвращает булев результат – true, если переданный ей параметр представляет собой числовые данные любого типа (со знаком или без знака, целочисленные или с плавающей точкой) либо математическое выражение, которое возвращает допустимое числовое значение.

С помощью функций **is_int()** и **is_float()** можно определить, является число целым или дробным. Еще две проверки считаются более сложными: функции is_finite() и **is_infinite()** позволяют выполнить именно те проверки, на которые указывают их имена (является число конечным или бесконечным). Но, строго говоря, диапазон значений, на которые распространяются эти функции, не может включать актуальной бесконечности. Вместо этого используются пределы диапазона значений с плавающей точкой, допустимые в конкретной системе.

```
<?PHP

    var_dump(is_numeric(4));                // true
    var_dump(is_numeric(25 - 6));           // true
    var_dump(is_numeric("25"));             // true
    var_dump(is_numeric("25 - 6"));         // false

    var_dump(is_int(4));                    // true
    var_dump(is_int(4.2));                  // false
    var_dump(is_int("4"));                  //  false -
данная проверка строже, чем проверка с помощью функции
is_numeric()

    var_dump(is_float(4));                  // false
    var_dump(is_float(4.0));                // true
    var_dump(is_float(M_PI));           // true
    var_dump(is_float(M_PI));           // true

?>
```

Результат работы скрипта представлен на рис. 4.6.

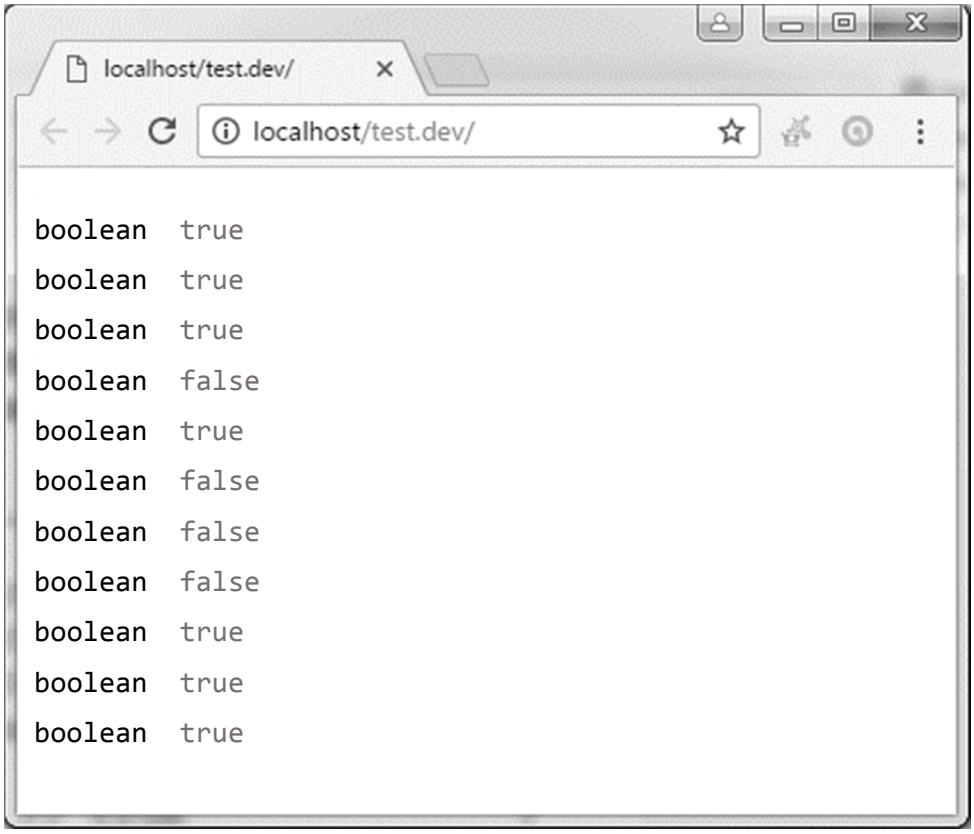

Рис. 4.6. Результат использования функций проверки формата чисел

С результатом работы скипта разберитесь самостоятельно.

4.5. Преобразование систем счисления

По умолчанию в языке PHP для прямого и обратного преобразования числовых значений из внешнего представления во внутреннее применяется основание системы счисления 10. Кроме того, можно сообщить интерпретатору PHP, что во внешнем представлении используются восьмеричные числа, заданные по основанию 8 (для этого перед числом необходимо ввести ведущий 0), или шестнадцатеричные числа, заданные по основанию 16 (для этого перед числом необходимо ввести префикс 0x).

Безусловно, после преобразования чисел из внешнего представления во внутреннее они хранятся в памяти в двоичном формате, а все основные арифметические и математические вычисления

осуществляются в самой операционной системе по основанию 2. Кроме того, в языке PHP предусмотрен ряд функций для преобразования чисел из одного основания системы счисления в другое. Общие сведения об этих функциях приведены в табл. 4.3.

Все функции преобразования систем счисления являются функциями специального назначения, преобразующими числа из одного конкретного основания в другое. Исключением является функция base_convert(), которая принимает произвольные параметры с обозначением начального и результирующего основания.

Таблица 4.3

Функции преобразования систем счисления

Функция	Описание
BinDec()	Принимает единственный строковый параметр, представляющий собой двоичное целое число (число по основанию 2) и возвращает строковое представление этого числа по основанию системы счисления 10
DecBin()	Аналогична BinDec(), но преобразует из основания системы счисления 10 в основание системы счисления 2
OctDec()	Аналогична BinDec(), но преобразует из основания системы счисления 8 в основание системы счисления 10
DecOct()	Аналогична BinDec(), но преобразует из основания системы счисления 10 в основание системы счисления 8
HexDec()	Аналогична BinDec(), но преобразует из основания системы счисления 16 в основание системы счисления 10
DecHex()	Аналогична BinDec(), но преобразует из основания системы счисления 10 в основание системы счисления 16
base_convert()	Принимает строковый параметр (представляющий целое число, которое подлежит преобразованию) и два целочисленных параметра (исходное и желаемое основание). Возвращает строку, представляющую преобразованное число. В этой строке цифры старше 9 (от 10 до 35) представлены символами *a–z*. И исходное, и желаемые основания должны находиться в пределах 2–36

```
<?PHP
    var_dump (bindec('1011'));
    var_dump (hexdec('FF'));
    var_dump (decbin('101'));
    var_dump (dechex('101'));
    var_dump (decoct('101'));
    var_dump (base_convert('1011', 2,10));
    var_dump (base_convert('FF', 16,20));
?>
```

Результат работы скрипта представлен на рис. 4.7.

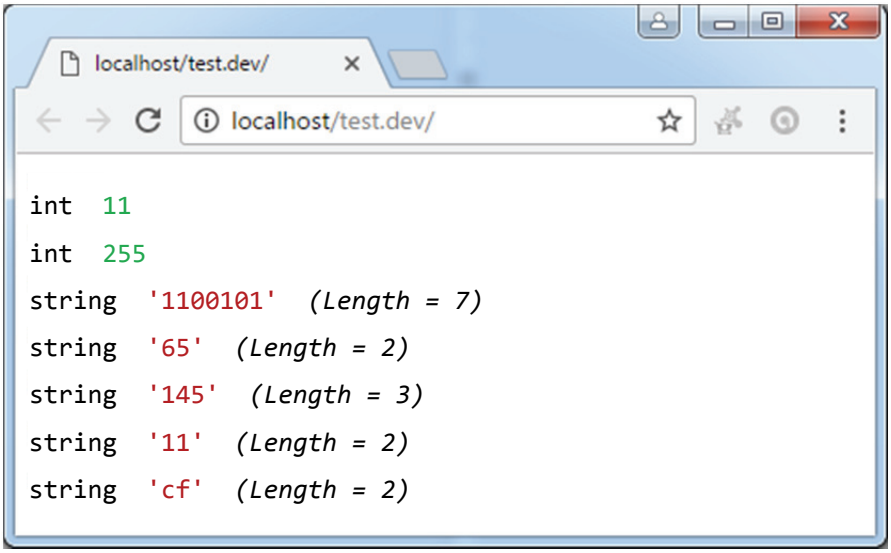

Рис. 4.7. Результат использования функций преобразования систем счисления

Обратите внимание на то, что все функции преобразования систем счисления принимают строковые параметры и возвращают строковые значения, но можно использовать десятичные числовые параметры и полагаться на правильное выполнение преобразования типа интерпретатором PHP. Иными словами, варианты вызова DecBin("1234") и DecBin(1234) приводят к получению одинакового результата.

4.6. Экспоненты и логарифмы

Язык PHP включает стандартные экспоненциальные и логарифмические функции двух разновидностей – для работы по основанию 10 и основанию *e* (приведены в табл. 4.4).

Таблица 4.4

Экспоненциальные функции

Функция	Описание
pow()	Принимает два числовых параметра и возвращает первый параметр, возведенный в степень, равную второму параметру. Значение выражения pow($*x*, $*y*) равно x^y
exp()	Принимает единственный параметр и возводит число *e* в степень, равную этому показателю степени. Значение выражения exp($*x*) равно e^x

Окончание табл. 4.4

Функция	Описание
log()	Функция натурального логарифма (ln). Принимает единственный параметр и возвращает его логарифм по основанию *e*
log10()	Принимает единственный параметр и возвращает его десятичный логарифм (lg)

В языке PHP 7-й версии также можно возводить в степень, используя **.

```
<?PHP
    $a=exp(1); echo 'a='.$a, '<br>';
    $a=pow(10,3); echo 'a='.$a, '<br>';
    $a=10**3; echo 'a='.$a, '<br>';
    $a=log10(100); echo 'a='.$a, '<br>';
?>
```

Результат работы скрипта представлен на рис. 4.8.

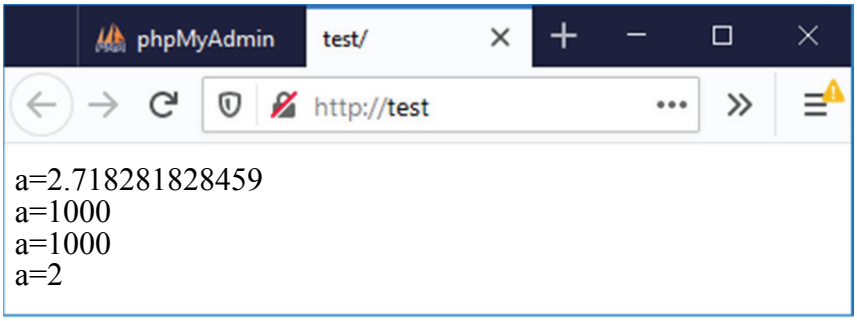

Рис. 4.8. Пример операций возведения в степень

Можно убедиться в том, что экспоненциальные и логарифмические функции с одним и тем же основанием являются обратными по отношению друг к другу, проведя проверку идентичности полученных результатов таким образом.

```
<?PHP
    $test = 449;
    $test = pow(10, exp(log(log10($test))));
    echo "test = $test";      // test_449 = 449
?>
```

Результат работы скрипта представлен на рис. 4.9.

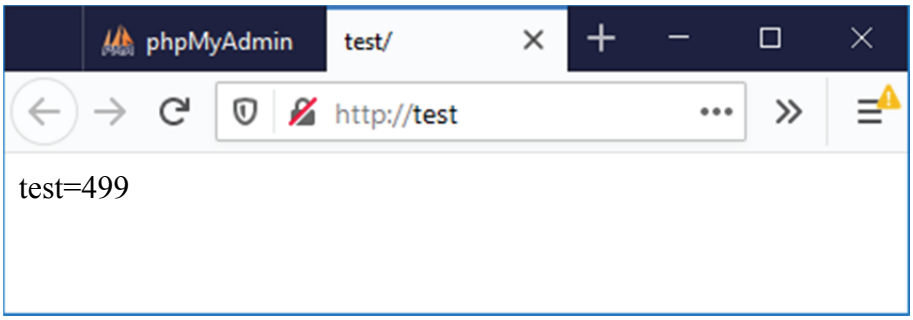

Рис. 4.9. Пример обратимости математических операций

В примере выводится имя переменной и ее значение, вычисленное с использованием функций, представленных в табл. 4.4.

4.7. Тригонометрические функции

В языке PHP предусмотрен стандартный набор основных тригонометрических функций, общие сведения о которых приведены в табл. 4.5.

Таблица 4.5

Тригонометрические функции

Функция	Описание
pi()	Не принимает параметров и возвращает приближенное значение числа π (3.1415926535898). Может использоваться как взаимозаменяемая с константой M_PI
sin()	Принимает числовой параметр в радианах и возвращает синус параметра в виде числа с плавающей точкой двойной точности
cos()	Принимает числовой параметр в радианах и возвращает косинус параметра в виде числа с плавающей точкой двойной точности
tan()	Принимает числовой параметр в радианах и возвращает тангенс параметра в виде числа с плавающей точкой двойной точности
asin()	Принимает числовой параметр и возвращает арксинус параметра в радианах. Входные данные должны находиться в пределах от –1 до 1 (получение функцией входных данных, выходящих за пределы этого диапазона, приводит к получению результата NAN). Результаты находятся в диапазоне от –π/2 до π/2
acos()	Принимает числовой параметр и возвращает арккосинус параметра в радианах. Входные данные должны находиться в пределах от –1 до 1 (получение функцией входных данных, выходящих за пределы этого диапазона, приводит к получению результата NAN. Результаты находятся в диапазоне от 0 до π

Окончание табл. 4.5

Функция	Описание
atan()	Принимает числовой параметр и возвращает арктангенс параметра в радианах. Результаты находятся в диапазоне от –π/2 до π/2
deg2rad()	Преобразует значение из градусов в радианы
rad2deg()	Преобразует значение из радианов в градусы

Примеры использования функций будут преведены дальше в тексте.

Практическая часть

Пример 1

В скрипте, представленном ниже, выполняется вычисление расстояния между двумя городами по их географическим координатам. При расчете учтена шарообразная форма планеты Земля. Расчет проводится с использованием тригонометрических функций. Следует обратить внимание на то, что аргументы тригонометрических функций должны быть в радианах. Поэтому для перевода градусов в радианы использована функция dec2rad(). Для вывода вычисленного значения расстояния с точностью до 10 м используем функцию sprintf(). Код скрипта сохраним в файле rmt.php.

```
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <h3 align="center">Расстояние между городами</h3>
    <p>Берлин (широта &phi;<sub>1</sub> = 52,48&deg;,
долгота &lambda;<sub>1</sub> = 13,05&deg;)<br />
      Варшава (широта&phi;<sub>2</sub> = 52,22&deg;,
долгота &lambda;<sub>2</sub> = 21,01&deg;)
    </p>
    <?PHP
      $φA = 52.48; $λA = 13.05;
      $φB = 52.22; $λB = 21.01;
      # преобразование грудусов в радианы
      $φArad = deg2rad($φA); $λArad = deg2rad($λA);
```

```
      $φBrad = deg2rad($φB); $λBrad = deg2rad($λB);
      # вычисление косинусов и синусов широт и разницы
долгот
      $cA = cos($φArad);
      $cB = cos($φBrad);
      $sA = sin($φArad);
      $sB = sin($φBrad);
      $delta = $λArad - $λBrad;
      $cdelta =cos($delta);
      $sdelta =sin($delta);
      # вычисление расстояния
      $a = pow($cB*$sdelta,2);
      $a = $a + pow($cA*$sB-$sA*$cB*$cdelta,2);
      $a = sqrt($a);
      $b = $sA*$sB+$cA*$cB*$cdelta;
      $ad = atan2($a, $b);
      $s = $ad * 6372.795;
      $s = sprintf ("%01.2f", $s);
      echo "Расстояние между Берлином и Варшавой $s
км";
    ?>
  </body>
</html>
```

Результат выполнения скрипта представлен на рис. 4.10.

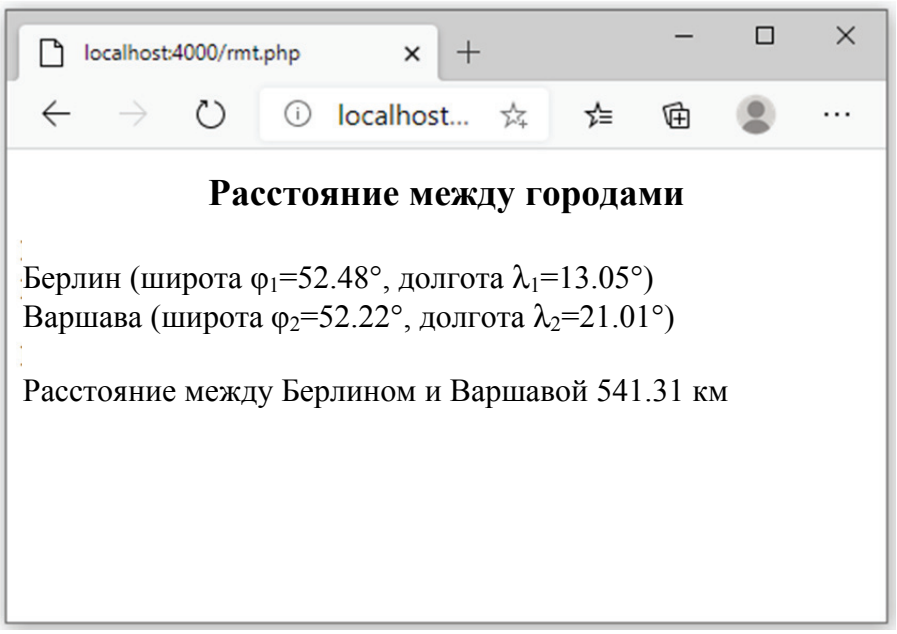

Рис. 4.10. Пример использования таблицы
и функций случайных чисел

Выполните расчет расстояния между городами, предположив, что планета Земля имеет плоскую форму, а не форму шара. Сравните полученные результаты и сделайте выводы.

Пример 2

В следующем скрипте приведен пример нахождения корней квадратного уравнения. В первых строчках кода записаны исходные данные четырех уравнений. Для нахождения корней уравнений используем тернарный оператор, который подробно будет рассмотрен в гл. 7. Скрипт сохраним в файле kvadrat.php

```
<?php
  #варианты уравнений
  #$a=1; $b=-2; $c=-3;
  #$a=-1; $b=-2; $c=15;
  #$a=1; $b=12; $c=36;
  $a=5; $b=3; $c=7;

  echo "Уравнение:
$a<i>x</i><sup>2</sup>+$b<i>x</i>+$c=0<br />";

  $D = $b**2-4*$a*$c;
  echo "Дискриминант: $D<br />";

  $x2 = ($D < 0) || ($D == 0) ? "нет корня" : (-$b-
sqrt($D))/2/$a;
  $x2 = ($D > 0) ? (-$b-sqrt($D))/2/$a : $x2;
  $x1 = ($D > 0) ? (-$b+sqrt($D))/2/$a:"нет корня";
  $x1 = ($D == 0) ? (-$b+sqrt($D))/2/$a : $x1;
  echo "Корни: x<sub>1</sub>=$x1  x<sub>2</sub>=$x2";
?>
```

Результат выполнения скрипта представлен на рис. 4.11.

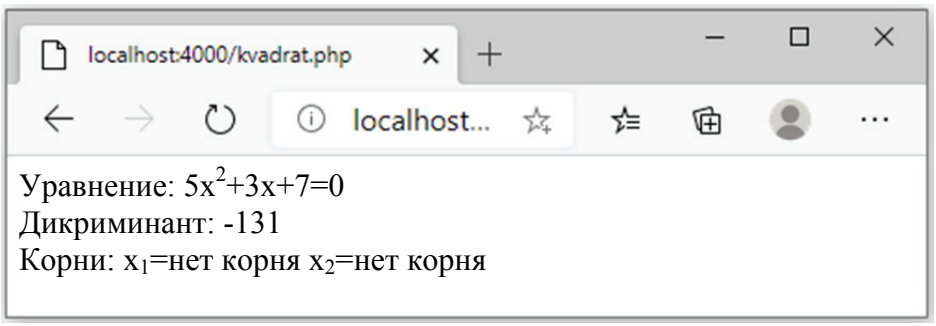

Рис. 4.11. Пример скрипта решения квадратного уравнения

Для решения этой задачи, возможно, более привычным будет использование оператора проверки условия (*if-elseif*), о котором будет рассказано подробно в гл. 7.

Лабораторная работа № 4

Задания на лабораторную работу будут носить похожий на рассмотренные примеры характер и выдаваться индивидульно преподавателем. Напомним, что выполнение и демострация всех лабораторных работ предполагается в пределах структуры, представленной в лабораторной работе № 1.

ГЛАВА 5

РАБОТА С СИМВОЛЬНЫМИ ПЕРЕМЕННЫМИ

5.1. Обработка строковых переменных (строк)

Если строка определяется в двойных кавычках либо при помощи heredoc, переменные внутри нее обрабатываются.

Существует синтаксис, дающий возможность обработки переменной, значения массива (array) или свойства объекта (object). Для этого используются { }.

Если интерпретатор встречает знак доллара ($), он захватывает так много символов, сколько возможно, чтобы сформировать правильное имя переменной. Для точного определения конца имени необходимо заключить имя переменной в фигурные скобки.

```
<?PHP
    $beer = 'Heineken';
    echo "He drank some $beers";    // не работает, 's'
это верный символ для имени переменной
    echo "<br>";
    echo "He drank some ${beer}s"; // работает
    echo "<br>";
    echo "He drank some {$beer}s"; // работает
?>
```

Результат работы скрипта представлен на рис. 5.1.

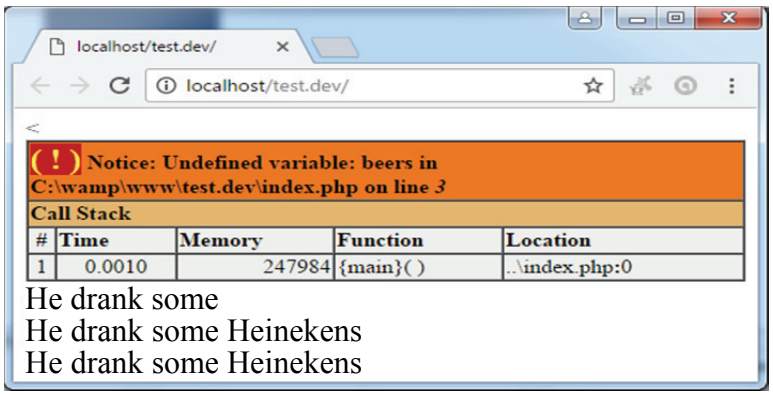

Рис. 5.1. Результат использования {} при выводе строк

Точно также могут быть обработаны элемент массива (array) или свойство объекта (object). В индексах массива закрывающая квадратная скобка (]) обозначает конец определения индекса.

```
<?PHP
    $great = 'fantastic';
    // Работает, выведет: b is fantastic
    echo "This is {$great}_ok";
    echo "<br>";
    echo "This is ${great}_ok";
    echo "<br>";
    echo "This is $great"."_ok";
    echo "<br>";
    // Пример работы с массивом
    $arr[0][0]="00";
    $arr[1][0]="10";
    $arr[0][1]="01";
    $arr[1][1]="11";
    var_dump ($arr);
    echo "Это работает: {$arr[1][1]}";
    echo "<br>";
    echo "Это не работает: $arr[1][1]";
?>
```

Результат работы скрипта представлен на рис. 5.2.

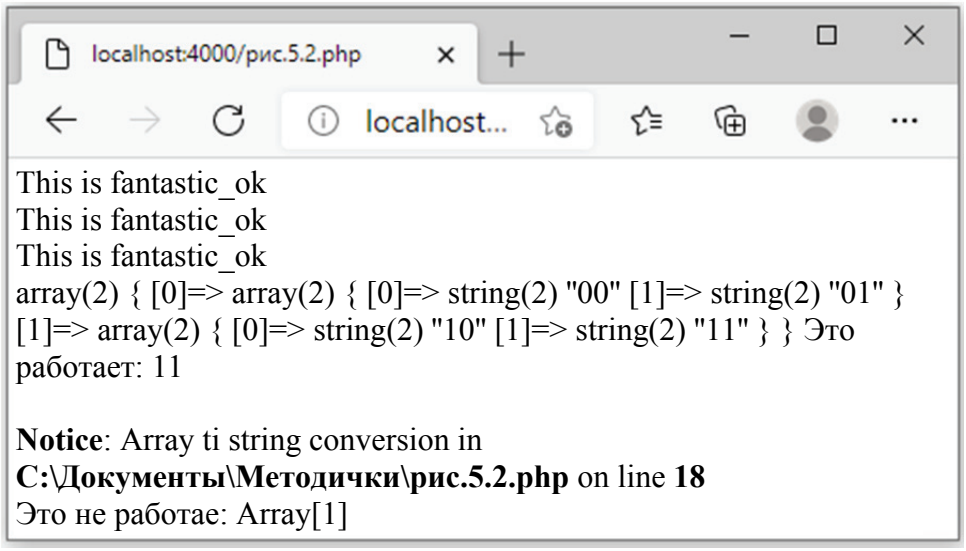

Рис. 5.2. Результат использования {} при выводе массивов

Результаты работы скрипта проанализируйте самостоятельно.

5.2. Доступ к символу в строке и его изменение

Символы в строках можно использовать и модифицировать, определив их смещение относительно начала строки, начиная с нуля, в фигурных (или крадратных, как элемент массива) скобках после строки. Приведем примеры.

```
<?PHP

    // Получение первого символа строки
    $str = 'Это тест.';
    $first = $str{0};
    echo $first,"<br>";

    // Получение третьего символа строки
    $first = $str{0};
    $third = $str{2};
    echo $third,"<br>";

    // Получение последнего символа строки
    $str = 'Это все еще тест.';
    $last = $str{strlen($str)-1};
    echo $last,"<br>";

    // Изменение последнего символа строки
    $str = 'Посмотри на море';
    $str{strlen($str)-1} = 'я';
    echo $str,"<br>";

?>
```

Результат работы скрипта представлен на рис. 5.3.

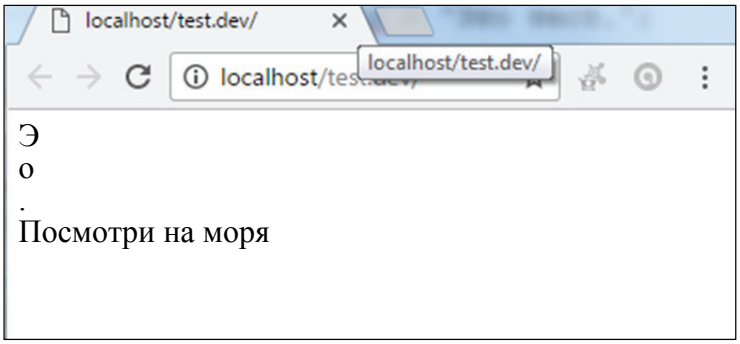

Рис. 5.3. Пример доступа к отдельным элементам строки

Результаты работы скрипта проанализируйте самостоятельно.

5.3. Строковые функции и операторы

5.3.1. Строковые операторы конкатенации. В различных языках программирования используются операторы конкатенации (объединения) строк. Например, в Pascal применяется оператор «+». Использование в PHP оператора «+» для конкатенации строк некорректно: если строки содержат числа, то вместо объединения строк будет выполнена операция сложения двух чисел. В PHP есть два оператора, выполняющих конкатенацию.

Первый – оператор конкатенации ('.'), который возвращает объединение левого и правого аргумента.

Второй – оператор присвоения с конкатенацией ('.='), который присоединяет правый аргумент к левому.

Приведем конкретный пример.

```
<?PHP
    $a = "Hello ";
    $b = $a."World!"; // $b содержит строку "Hello
World!" - Это конкатенация
    echo $b."<br>";

    $a = "Hello ";
    $a .= "World!";       // $a содержит строку "Hello
World!" - Это присвоение с конкатенацией
    echo $a;
?>
```

Результат работы скрипта представлен на рис. 5.4.

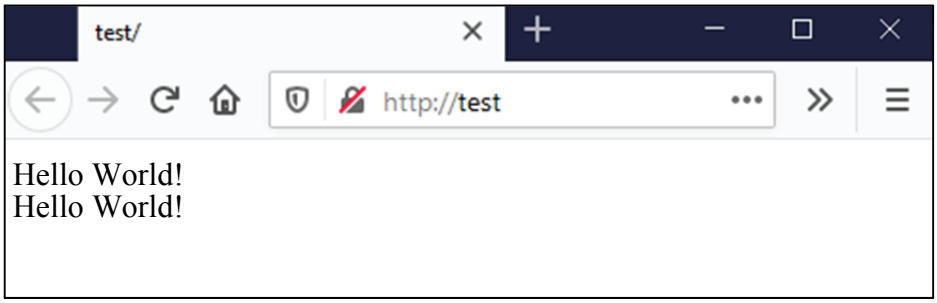

Рис. 5.4. Пример конкатенации строк

Результаты работы скрипта проанализируйте самостоятельно.

5.3.2. Операторы сравнения строк. Для сравнения строк не рекомендуется использовать операторы сравнения == и !=, поскольку они требуют преобразования типов.

```
<?PHP
    $x=0;
    $y=1;
    if ($x == "") echo "<p>x - пустая строка</p>";
    if ($y == "") echo "<p>y - пустая строка</p>";
?>
```

Результат работы скрипта представлен на рис. 5.5.

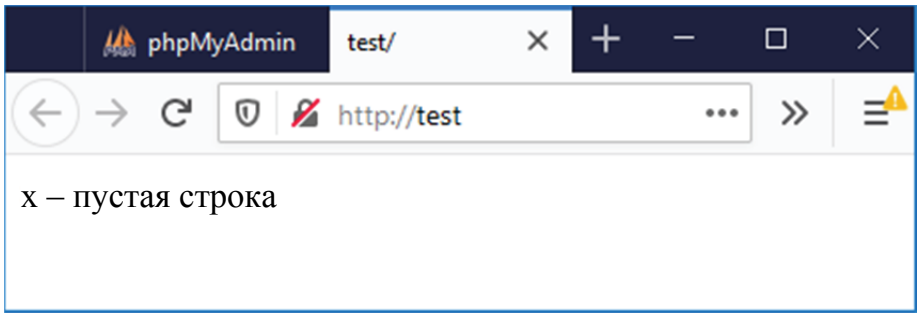

Рис. 5.5. Пример сравнения строк и чисел

Данный скрипт сообщает нам, что $x – пустая строка. Это связано с тем, что пустая строка ("") трактуется прежде всего как 0, а только затем – как «пусто». В PHP операнды сравниваются как строки только в том случае, если оба они – строки. В противном случае они сравниваются как числа. При этом любая строка, которую PHP не удается перевести в число (в том числе и пустая строка), будет восприниматься как 0.

Примеры сравнения строк:

```
<?PHP

    $x="Строка";
    $y="Строка";
    $z="Строчка";

    if ($x == $z) echo "<p>Строка X равна строке Z</p>";
    if ($x == $y) echo "<p>Строка X равна строке Y</p>";
    if ($x != $z) echo "<p>Строка X НЕ равна строке Z</p>";

?>
```

Результат работы скрипта представлен на рис. 5.6.

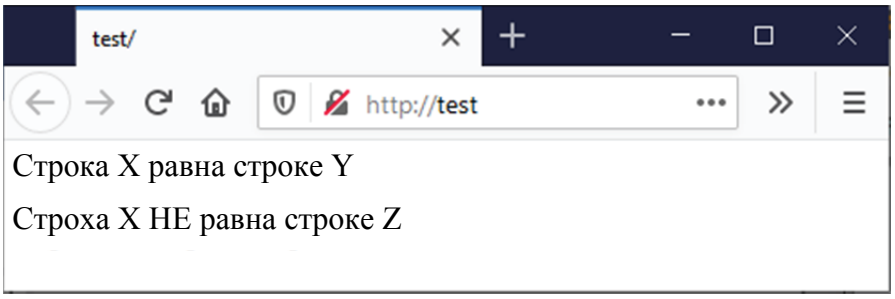

Рис. 5.6. Пример сравнения строк

Для избежания путаницы и преобразования типов рекомендуется пользоваться оператором эквивалентности при сравнении строк. Оператор эквивалентности позволяет всегда корректно сравнивать строки, поскольку сравнивает величины и по значению, и по типу.

```
<?PHP
    $x="Строка";
    $y="Строка";
    $z="Строчка";
    if ($x === $z) echo "<p>Строка X равна строке
Z</p>";
    if ($x === $y) echo "<p>Строка X равна строке
Y</p>";
    if ($x !== $z) echo "<p>Строка X НЕ равна строке
Z</p>";
?>
```

Результат работы скрипта представлен на рис. 5.7.

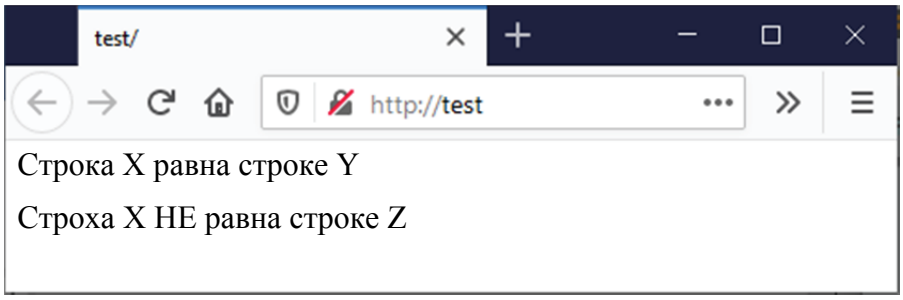

Рис. 5.7. Пример сравнения строк с использованием эквивалентности

Результаты работы скрипта проанализируйте самостоятельно.

5.4. Функции для работы со строками

Для работы со строками в PHP существует множество полезных функций. Кратко разберем некоторые из них.

5.4.1. Базовые строковые функции.

Функция strlen характеризуется следующим синтексисом.

```
strlen(string $st)
```

Одна из наиболее полезных функций. Возвращает просто длину строки, т. е. количество символов, содержащихся в $st. Строка может содержать любые символы, в том числе и с нулевым кодом. Пример:

```
<?PHP
    $x = "Hello!";
    echo strlen($x); // Выводит 6
?>
```

Результат работы скрипта представлен на рис. 5.8.

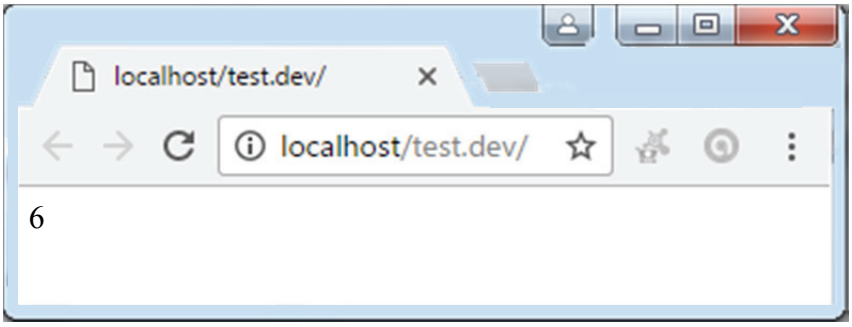

Рис. 5.8. Пример использования функции strlen()

Функция strpos характеризуется следующим синтаксисом:

```
strpos(string $where, string $what, int $fromwhere=0)
```

Данная функция пытается найти в строке $where подстроку (т. е. последовательность символов) $what и в случае успеха возвращает позицию (индекс) этой подстроки в строке. Необязательный параметр $fromwhere можно задавать, если поиск нужно ве-

сти не с начала строки, а с какой-то другой позиции. В этом случае следует эту позицию передать в $fromwhere. Если подстроку найти не удалось, функция возвращает false. Однако будьте внимательны, проверяя результат вызова strpos()на false – используйте для этого только оператор ===. Приведем пример.

```
<?PHP
echo strpos("Hello","el"); // Выводит 1
echo "<br>";
echo strpos("Hello! Hello! Hello! Hello! Hello!","el");
// Выводит 1
echo "<br>";
echo strpos("Hello! Hello! Hello! Hello! Hello!","el",
10); // Выводит 1
echo "<br>";
if (strpos("Norway","rwa") !== false) echo "Строка rwa
есть в Norway";
// При сравнении используйте операторы тождественных
сравнений (===) (!==), чтобы избежать проблем с опреде-
лением типов
?>
```

Результат работы скрипта представлен на рис. 5.9.

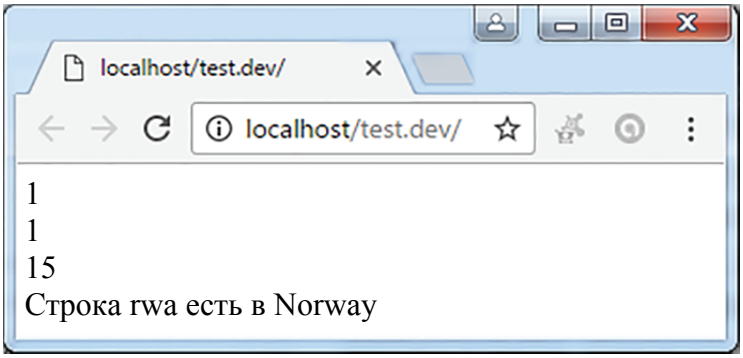

Рис. 5.9. Пример использования функции strops()

Функция substr характеризуется следующим синтаксисом:

```
substr(string $str, int $start [,int $length])
```

Данная функция очень востребуема. Ее назначение – возвращать участок строки $str начиная с позиции $start и длиной $length. Если $length не задана, то подразумевается подстрока от $start до

конца строки $str. Если $start больше, чем длина строки, или же значение $length равно нулю, то возвращается пустая подстрока. Однако эта функция может делать и еще довольно полезные вещи. К примеру, если мы передадим в $start отрицательное число, то будет считаться, что это число является индексом подстроки, но только отсчитываемым от конца $str (например, –1 означает «начиная с последнего символа строки»). Параметр $length, если он задан, тоже может быть отрицательным. В этом случае последним символом возвращенной подстроки будет символ из $str с индексом $length, определяемым от конца строки. Рассмотрим примеры.

```
<?PHP
    $str = "Programmer";
    echo substr($str,0,2); // Выводит Pr
    echo "<br>";
    echo substr($str,-4,3); // Выводит mme
    echo "<br>";
    echo substr($str,-5,-2); // Выводит amm
    echo "<br>";
    echo substr($str,-5,-4); // Выводит a
?>
```

Результат работы скрипта представлен на рис. 5.10.

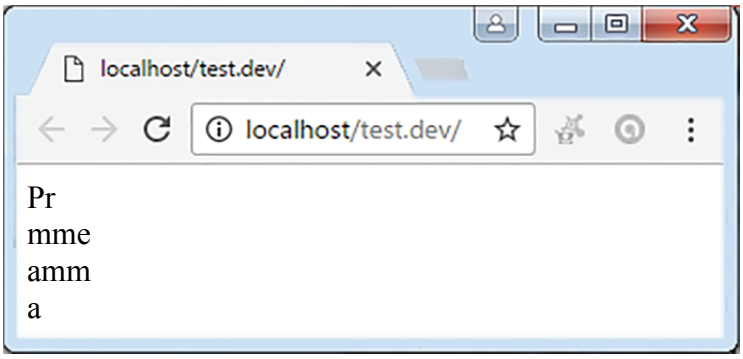

Рис. 5.10. Пример использования функции substr()

Функция strcmp характеризуется следующим синтаксисом:

```
    strcmp(string $str1, string $str2)
```

Сравнивает две строки посимвольно (точнее, побайтово) и возвращает: 0 – если строки полностью совпадают; –1 – если строка $str1 лексикографически меньше $str2; 1 – если, наоборот,

$str1 "больше" $str2. Так как сравнение идет побайтово, то регистр символов влияет на результаты сравнений.

Функция strcasecmp характеризуется следующим синтаксисом:

```
strcasecmp(string $str1, string $str2)
```

То же самое, что и strcmp(), только при работе не учитывается регистр букв. Например, с точки зрения этой функции «*ab*» и «*AB*» равны.

```
<?PHP
    $str1 = "Programmer";
    $str2 = "Programmer";
    $str3 = "Program";
    $str4 = "programmeR";
    echo strcmp($str1,$str2);
    echo "<br>";
    echo strcmp($str3,$str2);
    echo "<br>";
    echo strcmp($str2,$str3);
    echo "<br>";
    echo strcmp($str1,$str4);
    echo "<br>";
    echo "<br>";
    echo strcasecmp($str1,$str4); echo "<br>";
?>
```

Результат работы скрипта представлен на рис. 5.11.

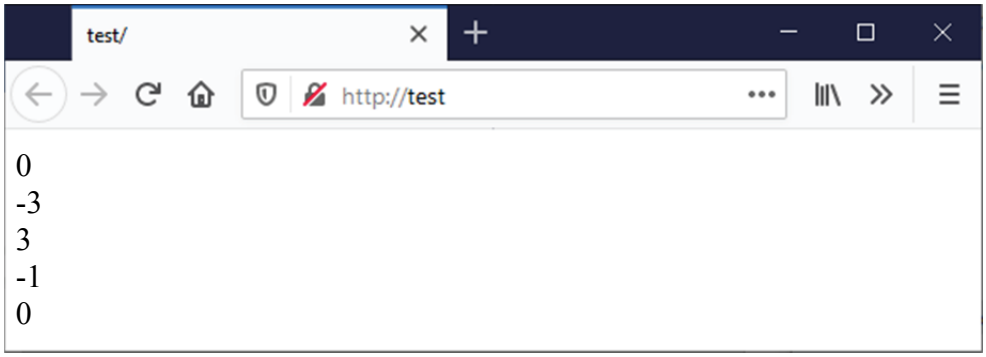

Рис. 5.11. Пример использования функции strcmp()

Функция strstr() находит первое вхождение подстроки.

```
strstr ( string $haystack , mixed $needle [, bool $befo
re_needle = FALSE ] ) : string
```

Данная функция возвращает подстроку строки *haystack* начиная с первого вхождения *needle* (и включая его) и до конца строки *haystack*.

Необходимо отметить, что данная функция учитывает регистр символов. Для поиска без учета регистра используйте *stristr()*.

Функция strrchr находит последнее вхождение символа в строке.

```
strrchr ( string $haystack , mixed $needle ) : string
```

Параметр *haystack* – это строка, в которой производится поиск. Параметр *needle* определяет процесс поиска. Если *needle* состоит более чем из одного символа, используется только первый символ. Это поведение отличается от *strstr()*.

Если параметр *needle* не является строкой, он преобразуется в целое число и трактуется как код символа. В зависимости от предполагаемого поведения параметр *needle* должен быть либо явно приведен к строке, либо должен быть выполнен явный вызов chr().

5.4.2. Функции для работы с блоками текста. Перечисленные ниже функции чаще всего оказываются полезны, если нужно проводить однотипные операции с многострочными блоками текста, заданными в строковых переменных.

Функция str_replace характеризуется следующим синтаксисом:

```
str_replace(string $from, string $to, string $str)
```

Заменяет в строке $str все вхождения подстроки $from (с учетом регистра) на $to и возвращает результат. Исходная строка, переданная третьим параметром, при этом не меняется.

```
<?PHP
$str="Выводим следующее предложение с новой строки \n";
$str1="Привет!";
echo $str, $str1;
echo"<br>","<br>";
$str2=str_replace("\n","<br>",$str);
echo $str2, $str1;
?>
```

Результат работы скрипта представлен на рис. 5.12.

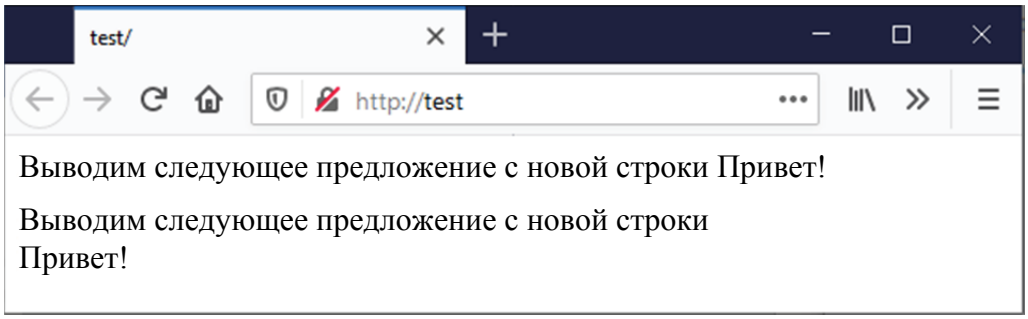

Рис. 5.12. Пример использования функции str_replace()

Приведем еще пример замены символов.

```
<?PHP
    $str="Hello! Hello! Hello!";
    echo $str;

    echo"<br>","<br>";

    $str1=str_replace("!","?",$str);
    echo $str1;
?>
```

Результат работы скрипта представлен на рис. 5.13.

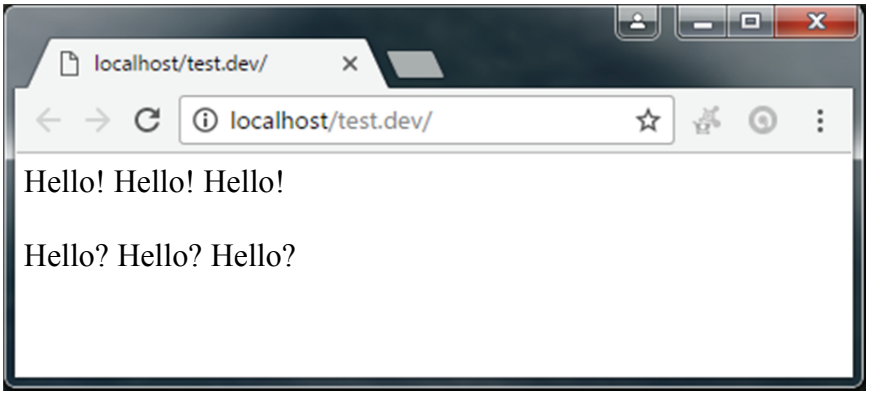

Рис. 5.13. Пример использования функции str_replace()

Данная функция производит лишь однократный проход по строке.

Функция nl2br имеет следующий синтаксис:

```
string nl2br(string $string)
```

Заменяет в строке все символы новой строки \n на
\n и возвращает результат. Исходная строка не изменяется. Обратите внимание на то, что символы \r, которые присутствуют в конце строки текстовых файлов Windows, этой функцией никак не учитываются, а потому остаются на старом месте.

Функция str_word_count возвращает информацию о словах, входящих в строку.

```
str_word_count ( string $string [, int $format =
0 [, string $charlist ]] ) : mixed
```

Подсчитывает количество слов, входящих в строку string. Если необязательный аргумент format не передан, возвращается целое число, равное количеству слов. В случае, если указан аргумент format, возвращается массив, содержимое которого зависит от значения format. Ниже описаны допустимые значения аргумента format и соответствующие им возвращаемые значения.

Для этой функции «слово» обозначает строку с алфавитными символами, зависящую от локали, которая также может содержать символы "'" и "–", но не может начинаться с них.

Функция WordWrap имеет следующий синтаксис:

```
WordWrap(string $str, int $width=75, string $break="\n")
```

Эта функция оказывается невероятно полезной, например, при форматировании текста письма перед автоматической отправкой его адресату при помощи mail(). Она разбивает блок текста $str на несколько строк, завершаемых символами $break, так, чтобы на одной строке было не более $width букв. Разбиение происходит по границе слова, так что текст остается читаемым. Возвращается получившаяся строка с символами перевода строки, заданными в $break. Пример использования:

```
<?PHP
    $str = "Это текст электронного письма, которое нуж-
но будет отправить адресату...";
    echo "$str <br>","<br>";
    // Разбиваем текст по 20 символов
    $str = WordWrap ($str, 20, "<br>");echo $str;
?>
```

Результат работы скрипта представлен на рис. 5.14.

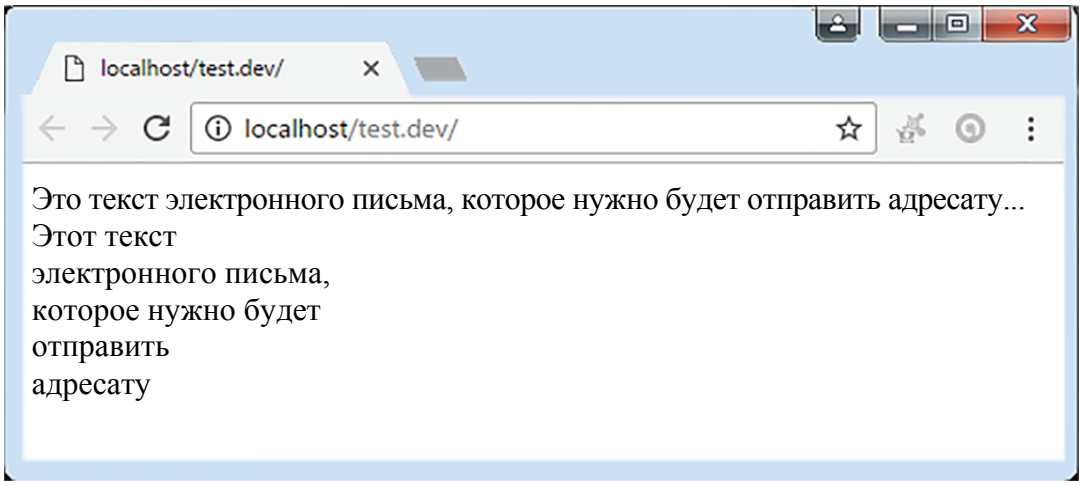

Рис. 5.14. Пример использования функции WordWrap()

Функция strip_tags характеризуется следующим синтаксисом:

```
strip_tags (string $str [, string $allowable_tags])
```

Еще одна полезная функция для работы со строками. Она удаляет из строки все теги и возвращает результат. В параметре $allowable_tags можно передать теги, которые не следует удалять из строки. Они должны перечисляться вплотную друг к другу.

```
$stripped = strip_tags ($str); // Удаляет все html -
теги из строки (текста)
$stripped = strip_tags($str, "<head><title>"); // Уда-
лит все html - теги, кроме html - тегов <head> и
<title>
```

Приведем пример.

```
<?PHP
    $str = "Это текст электронного письма,<br> которое
нужно будет <br> отправить адресату...";
    var_dump ($str);
    echo $str, "<br>";
    $stripped = strip_tags ($str); // Удаляет все html -
теги из строки (текста)
    var_dump ($stripped);
    echo $stripped, "<br>";
?>
```

Результат работы скрипта представлен на рис. 5.15.

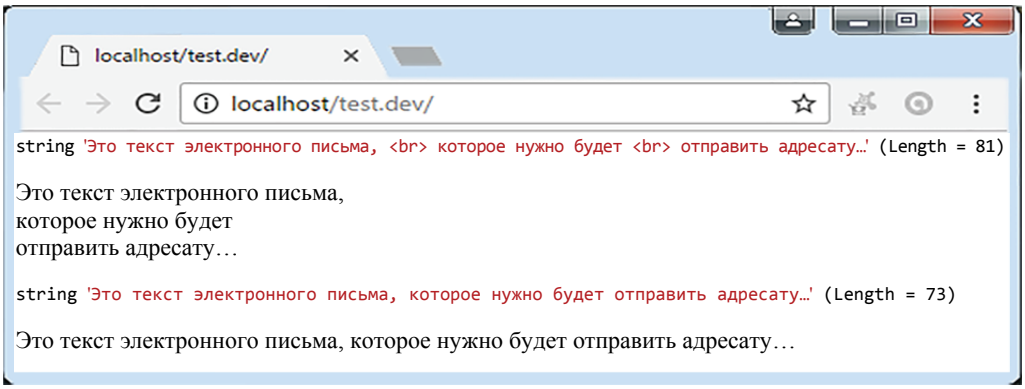

Рис. 5.15. Пример использования функции strip_tags()

Рассмотренные функции позволяют писать простой код при потребности в выполнении однотипных операций с многострочными блоками текста.

5.4.3. Функции для преобразования символьных переменных в массив и наоборот.

Функция explode () разбивает строку с помощью разделителя.

```
array explode ( string $delimiter , string $string [,
int $limit ] )
```

Возвращает массив строк, полученных разбиением строки string с использованием delimiter в качестве разделителя.

Список параметров:

– *Delimiter* – разделитель;

– *String* – входная строка;

– *Limit.* Если аргумент limit является положительным, возвращаемый массив будет содержать максимум limit элементов, при этом последний элемент будет содержать остаток строки string. Если параметр limit отрицателен, то будут возвращены все компоненты, кроме последних -limit. Если limit равен 0, то он расценивается как 1.

В результате возвращается массив (array) строк (string), созданный делением параметра string по границам, указанным параметром delimiter.

Если delimiter является пустой строкой (""), explode() возвращает FALSE. Если delimiter не содержится в string и используется отрицательный limit, то будет возвращен пустой массив (array), иначе будет возвращен массив, содержащий string.

```
<?PHP
    $text  = "part1 part2 part3 part4 part5 part6";
    $pieces = explode(" ", $text);
    echo $pieces[0]; // part1
    echo "<br>";
    echo $pieces[1]; // part2
    echo "<br><br>";
    var_dump ($pieces);
?>
```

Результат работы скрипта представлен на рис. 5.16.

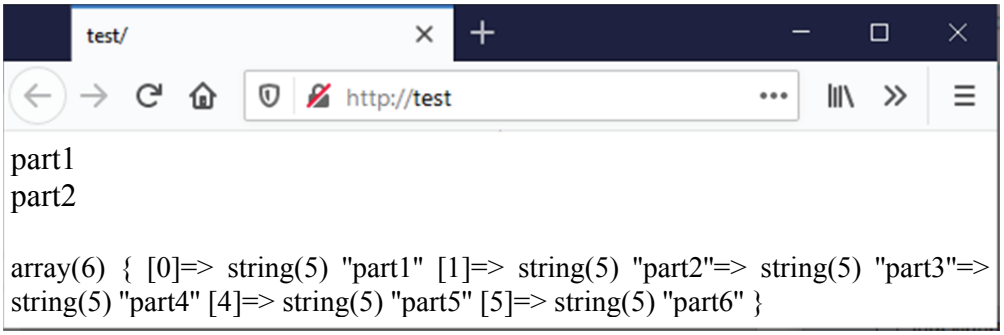

Рис. 5.16. Пример использования функции explode()

Можно привести и другой пример.

```
<?PHP
    $data = "foo:*:1023:1000::/home/foo:/bin/sh";
    list($user, $pass, $uid, $gid, $gecos, $home,
$shell) = explode(":", $data);
    echo $user,"<br>"; // foo
    echo $pass,"<br>"; // *
    echo $shell,"<br>"; // /bin/sh
?>
```

Результат работы скрипта представлен на рис. 5.17.

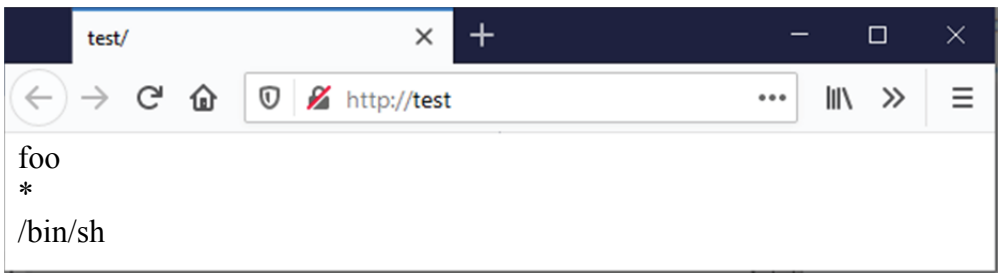

Рис. 5.17. Пример использования функции explode() с разделителем «:»

Функция Implode () объединяет элементы массива в строку с помощью glue.

```
string implode ( string $glue , array $pieces )
string implode ( array $pieces )
```

По историческим причинам, функции implode() можно передавать аргументы в любом порядке, однако для унификации с функцией explode() следует использовать документированный порядок аргументов.

Список параметров:

− *glue* – по умолчанию равен пустой строке;

− *pieces* – массив объединяемых строк.

В результате возвращает строку, содержащую строковое представление всех элементов массива в указанном порядке со строкой glue между каждым элементом.

```
<?PHP
    $array = array('имя', 'почта', 'телефон');
    $comma_separated = implode(",", $array);

    echo $comma_separated; // имя,почта,телефон
    echo "<br><br>";
    // Пустая строка при использовании пустого массива:
    var_dump(implode('hello',  array()));  //  string(0)
""
?>
```

Результат работы скрипта представлен на рис. 5.18.

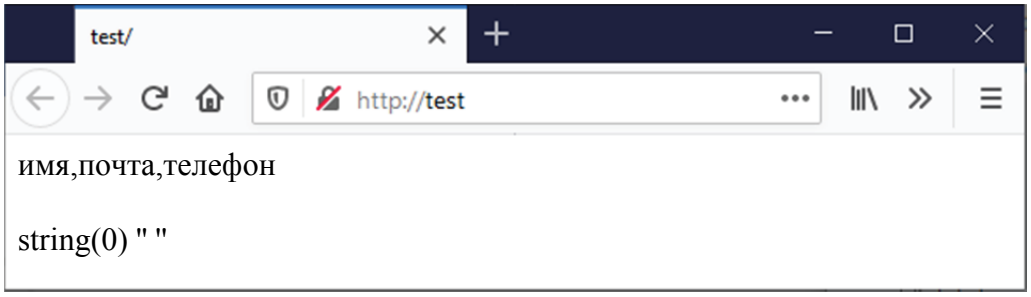

Рис. 5.18. Пример использования функции implode()

В предложенных примерах приведены возможности по работе с элементаит строк как элементами массива.

5.4.4. Функции для экранирования специальных символов.

Функция Quotemeta () экранирует специальные символы.

```
string quotemeta ( string $str )
```

Возвращает модифицированную строку, в которой перед каждым символом из следующего списка записывается символ «\»: . \ + * ? [^] ($).

В результате возвращает экранированную строку или FALSE, если в качестве параметра str была указана пустая строка.

```
<?PHP
    $str = "Hello world. (can you hear me?)";
    var_dump($str);
    $str1= quotemeta($str);
    var_dump($str1);
?>
```

Результат работы скрипта представлен на рис. 5.19.

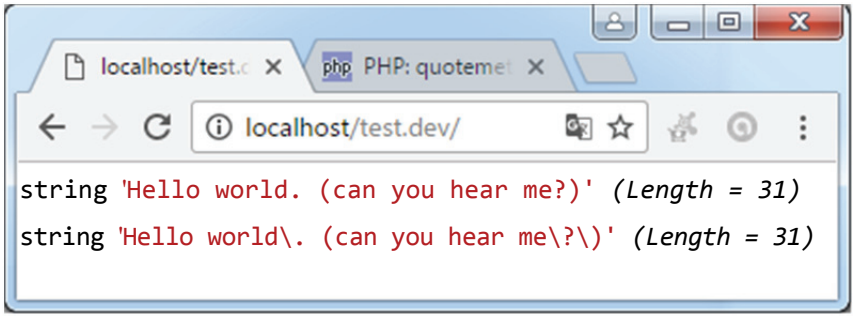

Рис. 5.19. Пример использования функции quotemeta()

Эта возможность может понадобиться в случае, если мы хотим продемонстрировать строку со специальными символами и результаты выполнения этой строки в браузере.

5.4.5. Функции для работы с отдельными символами.

Как и в других языках программирования, в PHP предоставляется возможность работать с символами строк отдельно. Обратиться к любому символу строки можно по его индексу.

```
$str = "PHP"; echo $str[0]; // Выводит 'P'
```

Функция chr характеризуется следующим синтаксисом:

```
chr(int $code)
```

Данная функция возвращает строку, состоящую из символа с кодом $code. Приведем простой пример, выводящий символ с ASCII кодом 75.

```
echo chr(75); //Выводит K
```

Функция ord характеризуется следующим синтаксисом:

```
ord($char)
```

Данная функция возвращает код символа $char. Вот пример вывода ASCII кода буквы «A».

```
echo ord('A'); // Выводит 65 – код буквы 'A'
```

Таблица ASCII. ASCII (American Standard Code for Information Interchange – стандартный американский код обмена информацией) – это код для представления символов в виде чисел, в котором каждому символу сопоставлено число от 0 до 127. В большинстве компьютеров код ASCII используется для представления текста, что позволяет передавать данные от одного компьютера на другой. Стандартный набор символов ASCII использует только 7 бит для каждого символа. Добавление 8-го разряда позволяет увеличить количество кодов таблицы ASCII до 255. Коды от 128 до 255 представляют собой расширение таблицы ASCII. Эти коды используются для кодирования символов национальных алфавитов, а также символов псевдографики, которые можно использовать, например, для оформления в тексте различных рамок и текстовых таблиц.

Коды таблицы ASCII представлены в прил. Б. Также можно написать скрипт, позволяющий просмотреть символы и их коды. Далее рассмотрим пример вывода на экран всех кодов ASCII и соответствующих им символам.

```
<?PHP
    for ($i=1; $i<=255; $i++){
        $symbol=chr($i);
        echo "код   $i  -  символ    $symbol","<br>";}
?>
```

Результат работы скрипта представлен на рис. 5.20.

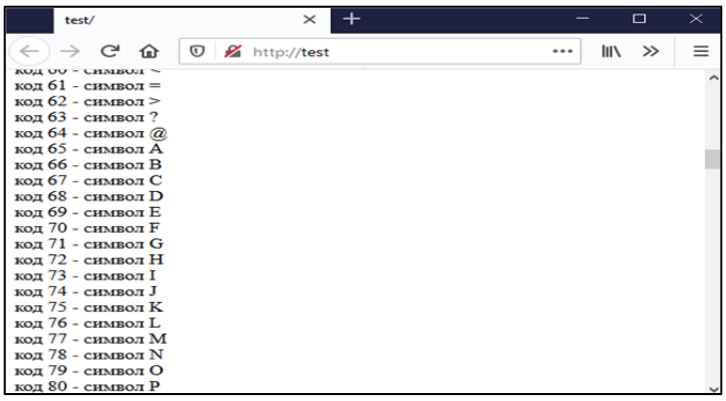

Рис. 5.20. Пример вывода кодов ASCII (фрагмент)

Представленные примеры продемонстрировали возможность обращения к отдельным символам строки для их изменения или получения информации не затрагивая остальные символы.

5.4.6. Функции удаления пробелов. Защитить себя от лишних пробелов чрезвычайно просто, и разработчики PHP предоставляют нам для этого ряд специализированных функций. Эти функции работают с молниеносной скоростью, а главное одинаково быстро, независимо от объема переданных им строк.

Функция trim характеризуется следующим синтаксисом:

```
    trim(string $str)
```

Возвращает копию $str, только с удаленными ведущими и концевыми пробельными символами. Под пробельными символами здесь и далее подразумевается: пробел " ", символ перевода строки \n, символ возврата каретки \r и символ табуляции \t. Например, вызов trim(" test\n ") вернет строку "test". Эта функция используется очень широко, ее надо применять везде, где есть хоть малейшее подозрение на наличие ошибочных пробелов.

```
<?PHP
    $a="Привет, Привет! Пока, Пока!\n";
    var_dump ($a);
    echo "<br";
    $b=trim($a);
    var_dump ($b);
?>
```

Результат работы скрипта представлен на рис. 5.21.

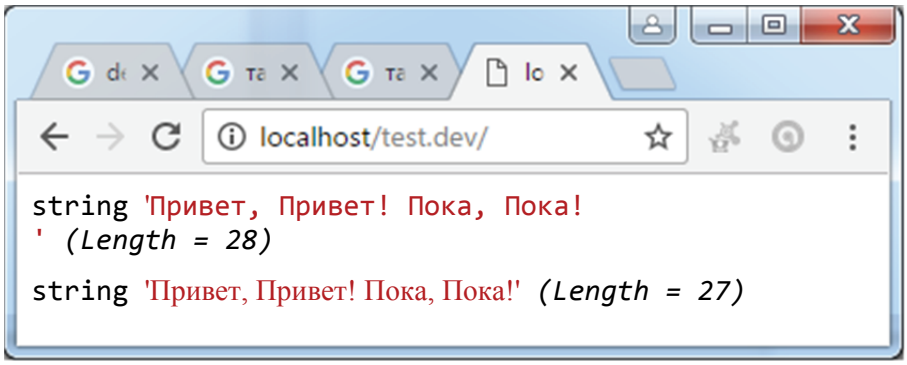

Рис. 5.21. Пример использования функции trim()

```
<?PHP
    $a="\tПривет, Привет!\t\t Пока, Пока!\t";
    var_dump ($a);
    echo "<br";
    $b=ltrim($a);
    var_dump ($b);
?>
```

Результат работы скрипта представлен на рис. 5.22.

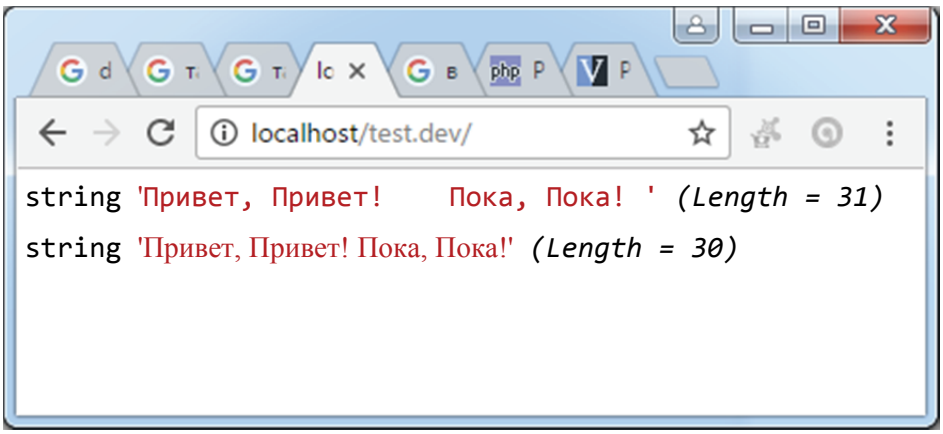

Рис. 5.22. Пример использования функции ltrim()

```
string trim ( string $str [, string $character_mask = "
\t\n\r\0\x0B" ] )
```

Эта функция возвращает строку *str* с удаленными из начала и конца строки пробелами. Если второй параметр не передан, *trim*() удаляет следующие символы:

— " " (ASCII *32* (*0x20*)) – обычный пробел;
— "\t" (ASCII *9* (*0x09*)) – символ табуляции;
— "\n" (ASCII *10* (*0x0A*)) – символ перевода строки;
— "\r" (ASCII *13* (*0x0D*)) – символ возврата каретки;
— "\0" (ASCII *0* (*0x00*)) – *NUL*-байт;
— "\x0B" (ASCII *11* (*0x0B*)) – вертикальная табуляция;

Str – обрезаемая строка (string);

character_mask – задает список символов для удаления. Является необязательным аргументом. Просто перечислите все символы, которые вы хотите удалить. Можно указать конструкцию .. для обозначения диапазона символов.

```
<?PHP
    $text   = "\t\tThese are a few words :) ...  ";
    $binary = "\x09Example string\x0A";
    $hello  = "Hello World";
    var_dump($text, $binary, $hello);

    print "\n";

    $trimmed = trim($text);
    var_dump($trimmed);

    $trimmed = trim($text, " \t.");
    var_dump($trimmed);

    $trimmed = trim($hello, "Hdle");
    var_dump($trimmed);

    $trimmed = trim($hello, 'HdWr');
    var_dump($trimmed);

    // удаляем управляющие ASCII-символы с начала и
конца $binary
    // (от 0 до 31 включительно)
    $clean = trim($binary, "\x00..\x1F");
    var_dump($clean);
?>
```

Результат работы скрипта представлен на рис. 5.23.

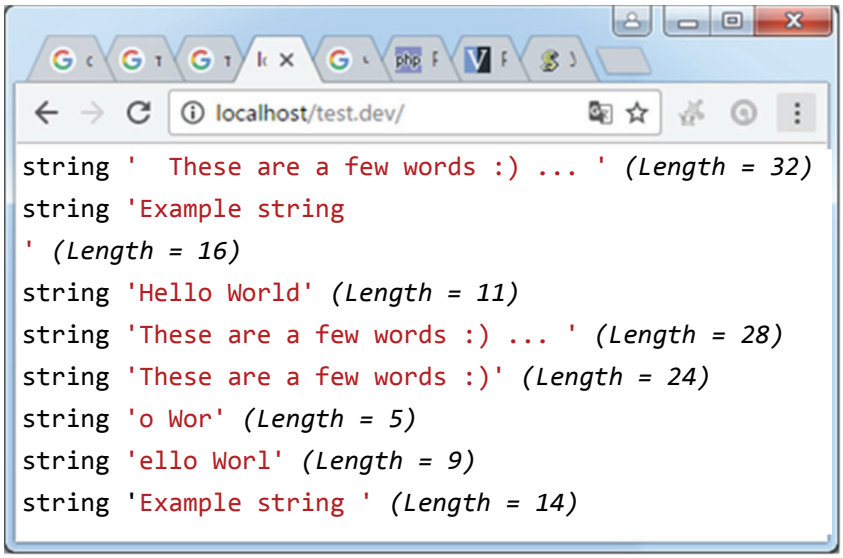

Рис. 5.23. Пример использования функции trim()
с дополнительным параметром

Функция ltrim характеризуется следующим синтаксисом:

```
ltrim(string $st)
```

То же, что и trim(), только удаляет исключительно ведущие пробелы, а концевые не трогает. Используется гораздо реже.

Функция chop характеризуется следующим синтаксисом:

```
chop(string $st)
```

Удаляет только концевые пробелы, ведущие не трогает.

5.4.7. Функции преобразования символов. Web-программирование – одна из тех областей, в которых постоянно приходится манипулировать строками: разрывать их, добавлять и удалять пробелы, перекодировать в разные кодировки, наконец, URL-кодировать и декодировать. В PHP реализовать все эти действия вручную, используя только уже описанные примитивы, просто невозможно из соображений быстродействия. Поэтому-то и существуют подобные встроенные функции.

Функция Strtr имеет следующий синтаксис:

```
strtr(string $str, string $from, string $to)
```

Эта функция применяется не столь широко, но все-таки иногда она бывает довольно полезной. Она заменяет в строке $str все символы, встречающиеся в $from, на их «парные», т. е. расположенные в тех же позициях, что и во $from, из $to.

```
<?PHP
    $addr="ПриВеТ";
    echo $addr;
    echo "<br>";
    $addr=strtr($addr,"АБВГДЕЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ",
"абвгдеежзийклмнопрстуфхцчшщъыьэюя");
    echo $addr;
?>
```

Результат работы скрипта представлен на рис. 5.24.

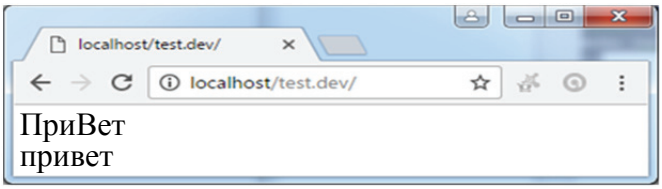

Рис. 5.24. Пример использования функции strtr()
для замены букв в верхнем регистре на буквы в нижнем регистре

```
<?PHP
    $addr="ГДЕ МЫ";
    echo $addr;
    echo "<br>";
    $addr=strtr($addr,"АБВГДЕМЫ", "ABVGDEMY");
    echo $addr;
?>
```

Результат работы скрипта представлен на рис. 5.25.

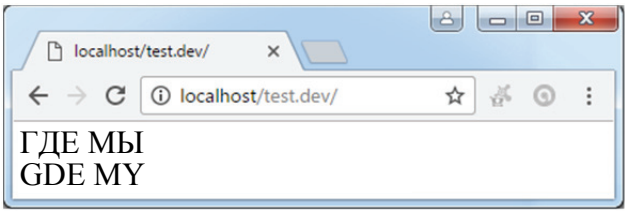

Рис. 5.25. Пример использования функции strtr()
для реализации смены алфавита

Следующие несколько функций предназначены для быстрого URL-кодирования и декодирования. URL-кодирование необходи-

мо для передачи данных через интернет. Например, такое кодирование целесообразно, если вы передаете русскоязычную информацию в качестве параметра скрипта. Также подобное кодирование можно выполнить и для файла, чтобы не возникало коллизий из-за отсутствия поддержки 8-битных кодировок некоторыми серверами. Данная группа функций преставлена в табл. 5.1.

Таблица 5.1

Функции URL-кодирования и декодирования

Наименование функции	Назначение
UrlEncode(string $str)	Функция URL кодирует строку $str и возвращает результат. Ее удобно применять, если вы, например, хотите динамически сформировать ссылку <a href=...> на какой-то сценарий, но не уверены, что его параметры содержат только алфавитно-цифровые символы. В этом случае воспользуйтесь функцией `echo "<a href=/script.php?param=". UrlEncode($UserData);` Теперь, даже если переменная $UserData включает символы =, & или даже пробелы, все равно сценарию будут переданы корректные данные
UrlDecode(string $st)	Производит URL-декодирование строки. В принципе используется значительно реже, чем UrlEncode(), потому что PHP и так умеет перекодировать входные данные автоматически
RawUrlEncode(string $st)	Почти полностью аналогична UrlEncode(), но только пробелы не преобразуются в «+», как это делается при передаче данных из формы, а воспринимаются как обычные неалфавитно-цифровые символы. Впрочем, этот метод не порождает никаких дополнительных несовместимостей в коде
RawUrlDecode(string $st)	Аналогична UrlDecode(), но не воспринимает «+» как пробел
HtmlSpecialChars(string $str)	Это функция, которая обычно используется в комбинации с echo. Основное ее назначение – гарантировать, что в выводимой строке ни один участок не будет воспринят как тег. Заменяет в строке некоторые символы (такие как амперсант, кавычки и знаки «больше» и «меньше») на их HTML-эквиваленты так, чтобы они выглядели на странице «самими собой». Самое типичное применение этой функции – формирование параметра value в различных элементах формы, чтобы не было никаких проблем с кавычками, или же вывод сообщения в гостевой книге, если вставлять теги пользователю запрещено

Окончание табл. 5.1

Наименование функции	Назначение
StripSlashes(string $str)	Заменяет в строке $str некоторые предваренные слешем символы на их однокодовые эквиваленты. Это относится к следующим символам: ", ', \ и никаким другим
AddSlashes(string $str)	Вставляет слеши только перед следующими символами: ', " и \. Функцию очень удобно использовать при вызове eval()(эта функция выполняет строку, переданную ей в параметрах, так, как будто имеет дело с небольшой PHP-программой

Пример применения этих функций будет представлен ниже.

5.4.8. Функции изменения регистра. Довольно часто приходится переводить какие-то строки, скажем, в верхний регистр, т. е. делать все прописные буквы в строке заглавными. В принципе, для этой цели можно было бы воспользоваться функцией strtr(), рассмотренной выше, но она все же будет работать не так быстро, как нам иногда хотелось бы. В PHP есть функции, которые предназначены специально для таких целей.

Функция strtolower характеризуется следующим синтаксисом:

```
strtolower(string $str)
```

Преобразует строку в нижний регистр. Возвращает результат перевода.

```
<?PHP
    $str = "Mary Had A Little Toy and She LOVED It
So";
    echo $str,"<br>";
    $str = strtolower($str);
    echo $str; // выводит: mary had a little lamb and
she loved it so
?>
```

Результат работы скрипта представлен на рис. 5.26.

Рис. 5.26. Пример использования функции strtolower()

Надо заметить, что при неправильной настройке локали (это набор правил по переводу символов из одного регистра в другой, переводу даты и времени, денежных единиц и т. д.) функция будет выдавать неправильные результаты при работе с буквами кириллицы. Возможно, в несложных программах, а также если нет уверенности в поддержке соответствующей локали операционной системой, проще будет воспользоваться «ручным» преобразованием символов, задействуя функцию strtr().

```
$st=strtr($st,      "АБВГДЕЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ",
"абвгдеежзийклмнопрстуфхцчшщъыьэюя");
```

Главное достоинство данного способа – то, что в случае проблем с кодировкой для восстановления работоспособности сценария придется всего лишь преобразовать его в ту же кодировку, в которой у вас хранятся документы на сервере.

Функция Strtoupper имеет следующий синтаксис:

```
strtoupper(string $str)
```

Переводит строку в верхний регистр. Возвращает результат преобразования. Эта функции также прекрасно работает со строками, составленными из латиницы, но с кириллицей может возникнуть все та же проблема кодировок.

5.4.9. Функции преобразования кодировок. Часто встречается ситуация, когда нам требуется преобразовать строку из одной кодировки кириллицы в другую. Например, в программе была кодировка windows, а стала – KOI8-R. Но строки-то остались по-прежнему в кодировке WIN-1251, а значит, для правильной работы с ними нужно их перекодировать в KOI8-R. Для этого и служит функция преобразования кодировок.

Функция convert_cyr_string имеет следующий синтаксис:

```
convert_cyr_string(string $str, char $from, char $to);
```

Функция переводит строку $str из кодировки $from в кодировку $to. Конечно, это имеет смысл только для строк, содержащих «русские» буквы, так как латиница во всех кодировках выглядит одинаково. Разумеется, кодировка $from должна совпадать с истинной кодировкой строки, иначе результат получится неверным. Значения $from и $to – один символ, определяющий кодировку: k – koi8-r, w – windows-1251, i – iso8859-5, a – x-cp866, d – x-cp866, m – x-mac-cyrillic.

Функция работает достаточно быстро, так что ее вполне можно применять, скажем, для перекодировки писем в нужную форму перед их отправкой по электронной почте.

5.4.10. Хеш-функции.

Функция md5 имеет следующий синтаксис:

```
md5(string $str)
```

Возвращает хеш-код строки $str, основанный на алгоритме корпорации RSA Data Security под названием "MD5 Message-Digest Algorithm". Хеш-код – это просто строка, практически уникальная для каждой из строк $str, т. е. вероятность того, что две разные строки, переданные в $str, дадут нам одинаковый хеш-код, стремится к нулю. Если длина строки $str может достигать нескольких тысяч символов, то ее MD5-код занимает максимум 32 символа. Хеш-код на основе алгоритма MD5 используется, например, для проверки паролей на истинность. Пусть, к примеру, у нас есть система со многими пользователями, каждый из которых имеет свой пароль. Можно, конечно, хранить все эти пароли в обычном виде или зашифровать их каким-нибудь способом, но тогда велика вероятность того, что в один прекрасный день этот файл с паролями у вас украдут. Сделаем так: в файле паролей будем хранить не сами пароли, а их (MD5) хеш-коды. При попытке какого-либо пользователя войти в систему мы вычислим хеш-код только что введенного им пароля и сравним его с тем, который записан у нас в базе данных.

Конечно, при вычислении хеш-кода какая-то часть информации о строке $str безвозвратно теряется. И именно это позволяет

не опасаться, что злоумышленник, получивший файл паролей, сможет его когда-нибудь расшифровать.

Пример использования алгоритма хеширования MD5:

```
<?PHP
  $pass_a = "MySecret";
  $pass_b = "MySecret";
  // Выводим хеш-код строки MySecret ($pass_a) - исход-
ный пароль
  echo "<b>Хеш-код исходного пароля '$pass_a':</b><b
style=\"color:green\">".md5($pass_a)."</b><br>";
  // Выводим хеш-код строки MySecret ($pass_b) - вери-
фицируемый пароль
  echo "<b>Хеш-код верифицируемого пароля '$pass_b':</b><b
style=\"color:green\">".md5($pass_b)."</b><br>";
  // Сравниваем хеш-коды MD5 исходного и верифицируемо-
го пароля
  echo "<h3>Проверяем истинность введенного паро-
ля:</h3>";
  if (md5($pass_a)===md5($pass_b)) echo "<h3
style=\"color:green\">Пароль верный! (Хеш-коды совпадают)</h3>";
     else echo "<h3 style=\"color:red\">Пароль
неверный! (Хеш-коды не совпадают)</h3>"
     // В данной ситуации выводит: Пароль верный!
(Хеш-коды совпадают)
?>
```

Результат работы скрипта представлен на рис. 5.27.

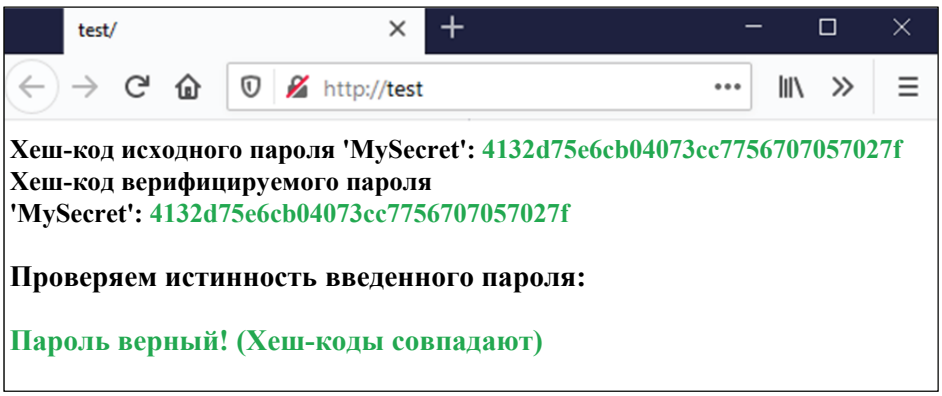

Рис. 5.27. Пример использования функций хеширования

Функция sha1 имеет следующий синтаксис:

```
string sha1 (string $str [, bool $raw_output = false] )
```

Возвращает SHA1-хеш строки str, вычисленный по алгоритму US Secure Hash Algorithm 1. Если необязательный аргумент raw_output имеет значение TRUE, хеш возвращается в виде бинарной строки из 20 символов, иначе он будет возвращен в виде 40-символьного шестнадцатеричного числа.

```
<?PHP
    $str = 'яблоко';
    echo sha1($str);
?>
```

Результат работы скрипта представлен на рис. 5.28.

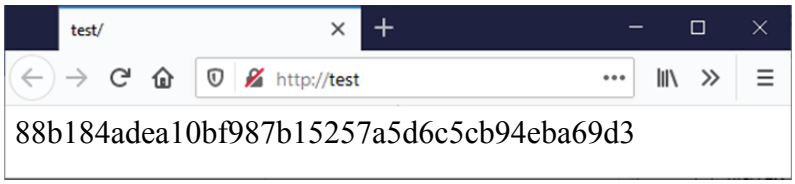

Рис. 5.28. Пример использования функции sha1()

Функция md5_file возвращает MD5-хеш файла.

```
string md5_file ( string $filename [, bool $raw_output
= false ] )
```

Вычисляет MD5-хеш файла, имя которого задано аргументом filename, используя алгоритм MD5 RSA Data Security, Inc. и возвращает этот хеш. Хеш представляет собой 32-значное шестнадцатеричное число. Если raw_output имеет значение TRUE, то возвращается бинарная строка из 16 символов.

```
<?PHP
  $file = '1.txt';
  echo 'MD5-хеш файла ' . $file . ': ' . md5_file($file);
?>
```

Файл 1.txt представлен на рис. 5.29.

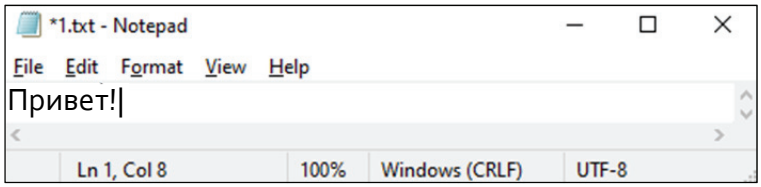

Рис. 5.29. Файл 1.txt

Результат работы скрипта представлен на рис. 5.30.

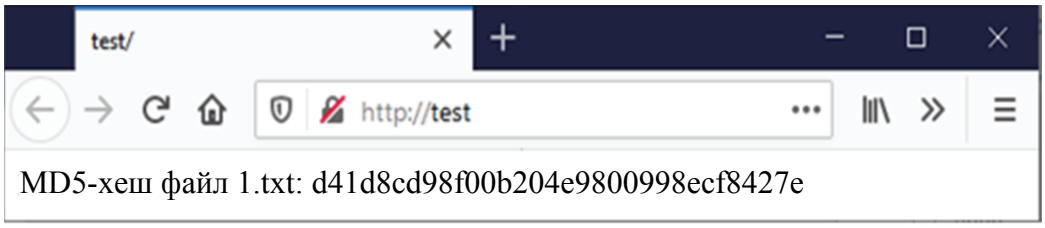

Рис. 5.30. Пример хеширования файла

Функция sha1_file возвращает SHA1-хеш файла.

```
string sha1_file ( string $filename [, bool $raw_output
 = false ] )
```

Вычисляет SHA1-хеш файла, имя которого задано аргументом filename, используя алгоритм US Secure Hash Algorithm 1 и возвращает этот хеш. Если raw_output установлен в TRUE, возвращает 20-символьный хеш в бинарном формате.

Функция crypt реализует необратимое хеширование строки:

```
crypt ( string $str [, string $salt ] ) : string
```

Функция crypt() возвращает хешированную строку, полученную с помощью стандартного алгоритма UNIX, основанного на DES или другого алгоритма, имеющегося в системе. Параметр salt является необязательным. Однако без salt функция crypt() создает слабый пароль. PHP 5.6 и новее вызывают ошибку E_NOTICE, если не использовать соль.

Функция password_hash() использует сложный хеш, генерирует сложную соль и применяет правильно количество раундов хеширования автоматически. Данная функция является простой оберткой над crypt() и совместима с существующими хешами паролей, поэтому и рекомендуется использование password_hash().

```
password_hash ( string $password , mixed $algo    [, array
$options ] ) : string
```

Ввиду того, что функция password_hash() совместима с функцией crypt(), хеши паролей, созданные crypt(), можно использовать с password_hash().

5.4.11. Функции сравнения строк по расстоянию.

Функция levenshtein() вычисляет расстояние Левенштейна между двумя строками.

```
int levenshtein ( string $str1 , string $str2 )
int levenshtein ( string $str1 , string $str2 , int $co
st_ins , int $cost_rep , int $cost_del )
```

Расстояние Левенштейна – это минимальное количество вставок, замен и удалений символов, необходимое для преобразования str1 в str2.

В простейшей форме функция принимает в качестве аргументов две строки и возвращает минимальное количество вставок, замен и удалений символов, необходимое для преобразования str1 в str2.

Второй вариант принимает три дополнительных аргумента, задающих стоимость операций вставки, замены и удаления. Этот вариант универсальнее первого, но не так эффективен.

Таким образом, список параметров функции следующий:

– *str1* – одна из строк, для которых вычисляется расстояние Левенштейна;

– *str2* – одна из строк, для которых вычисляется расстояние Левенштейна;

– *cost_ins* – определяет стоимость вставки;

– *cost_rep* – определяет стоимость замены;

– *cost_del* – определяет стоимость удаления.

Эта функция возвращает расстояние Левенштейна между двумя строками, или –1, если хотя бы одна из строк длиннее 255 символов.

```
<?PHP
// введенное слово с опечаткой
$input = 'carrrot';
// массив сверяемых слов
$words  = array('apple','pineapple','banana','orange',
    'radish','carrot','pea','bean','potato');
// кратчайшее расстояние пока еще не найдено
$shortest = -1;
// проходим по словам для нахождения самого близкого
варианта
```

```
foreach ($words as $word) {
    // вычисляем расстояние между входным словом и текущим
    $lev = levenshtein($input, $word);
    // проверяем полное совпадение
    if ($lev == 0) {
        // это ближайшее слово (точное совпадение)
        $closest = $word;
        $shortest = 0;

        // выходим из цикла - мы нашли точное совпадение
        break;
    }
    // если это расстояние меньше следующего наименьше-
го расстояния
    // ИЛИ если следующее самое короткое слово еще не
было найдено
    if ($lev <= $shortest || $shortest < 0)
    {
        // set the closest match, and shortest distance
        $closest  = $word;
        $shortest = $lev;
    }
}
echo "Вы ввели: $input","<br>";
if ($shortest == 0)
{
    echo "Найдено точное совпадение: $closest\n";
} else {
    echo "Вы не имели в виду: $closest?\n";
}
?>
```

Результат работы скрипта представлен на рис. 5.31.

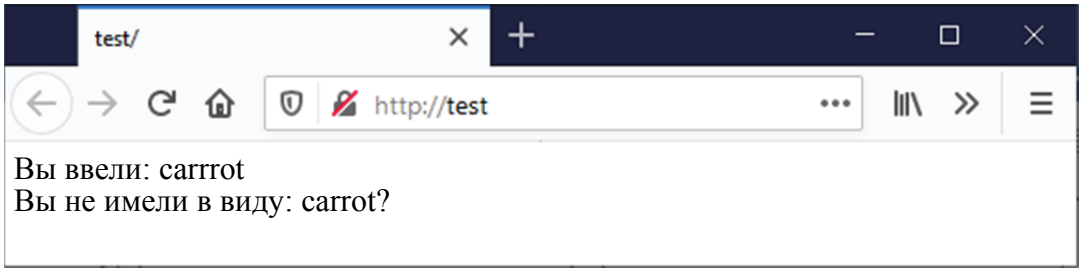

Рис. 5.31. Пример использования расстояния Левенштейна

Введя неверное значение, можно найти правильное.

5.5. Установка локали (локальных настроек)

Локалью будем называть совокупность локальных настроек системы, таких как формат даты и времени, язык, кодировка. Настройки локали сильно зависят от операционной системы. Для установки локали используется функция SetLocale().

```
SetLocale(string $category, string $locale)
```

Функция устанавливает текущую локаль, с которой будут работать функции преобразования регистра символов, вывода даты-времени и т. д. Вообще говоря, для каждой категории функций локаль определяется отдельно и выглядит по-разному. То, какую именно категорию функций затронет вызов SetLocale(), задается в параметре $category. Он может принимать следующие строковые значения:

− LC_CTYPE – активизирует указанную локаль для функций перевода в верхний / нижний регистры;

− LC_NUMERIC – активизирует локаль для функций форматирования дробных чисел (задает разделитель целой и дробной части в числах);

− LC_TIME – задает формат вывода даты и времени по умолчанию;

− LC_ALL – устанавливает все вышеперечисленные режимы.

Теперь о параметре $locale. Как известно, каждая локаль, установленная в системе, имеет свое уникальное имя, по которому к ней можно обратиться. Именно оно и фиксируется в этом параметре. Однако есть два важных исключения из правила. Во-первых, если величина $locale равна пустой строке "", то устанавливается та локаль, которая указана в глобальной переменной окружения с именем, совпадающим с именем категории $category (или LANG – она практически всегда присутствует в Unix). Во-вторых, если в этом параметре передается 0, то новая локаль не устанавливается, а просто возвращается имя текущей локали для указанного режима.

К сожалению, имена локалей задаются при настройке операционной системы, и для них, по-видимому, не существует стандартов. Выясните у своего хостинг-провайдера, как называются локали для разных кодировок русских символов. Но если следующий фрагмент работает у вашего хостинг-провайдера,

это не означает, что он заработает, например, под Windows: setlocale('LC_CTYPE','ru_SU.KOI8-R').

Здесь вызов устанавливает таблицу замены регистра букв в соответствии с кодировкой KOI8-R.

Необходимо отметить, что локаль в целом довольно непредсказуема и довольно плохо переносима между операционными системами, поэтому возможно лучше будет искать обходной путь, например, используя strtr(), а не рассчитывать на локаль.

5.6. Форматные преобразования строк

Переменные в строках PHP интерпретируются, поэтому практически всегда задача «смешивания» текста со значениями переменных не является проблемой. Например, мы можем спокойно написать что-то вроде

```
echo "Привет, $name! Вам $age лет.";
```

Язык PHP также поддерживает ряд функций, использующих такой же синтаксис, как в языке C. Бывают случаи, когда их применение дает наиболее красивое и лаконичное решение, хотя это и случается довольно редко.

Функция print. Выводит строку *print*. Не является «настоящей» функцией (это конструкция языка), поэтому заключать аргументы в скобки необязательно. Единственное отличие от *echo* в том, что *print* принимает только один аргумент.

```
int print ( string $arg )
```

Эта функция – аналог функции sprintf() в языке C. Она возвращает строку, составленную на основе строки форматирования, содержащей некоторые специальные символы, которые будут впоследствии заменены на значения соответствующих переменных из списка аргументов.

Рассмотрим пример:

```
<?PHP
    print("Привет мир!");
    print "<br> Привет мир!";
    print  "<br>Это займет\nнесколько строк. Переводы
строки тоже\nвыводятся";
    print "<br>Это займет<br>несколько строк. Переводы
строки тоже<br>выводятся";
```

```
    print "<br>Экранирование символов делается \"Так\".";

    // с print можно использовать переменные ...
    $a = "aaaaa";
    $b = "bbbbb";
    print "<br>это a=$a , b=$b";

    // ... и массивы
    $bar = array("value" => "AAA", "type" => "BBB");
    print "<br>это {$bar['value']} !";

    // При использовании одиночных кавычек выводится
имя переменной, а не значение
    print '<br>foo - это $a<br>';

    // Если вы не используете другие символы, можно вы-
вести просто значения переменных
    print $a;
?>
```

Результат работы скрипта представлен на рис. 5.32.

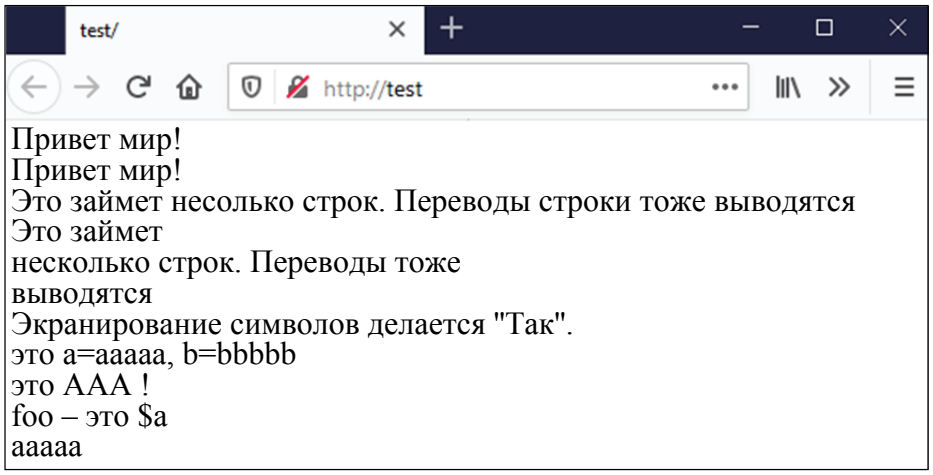

Рис. 5.32. Пример форматирования с использованием print

```
sprintf(string $format [, mixed args, ...])
```

Строка форматирования $format может включать в себя команды форматирования, предваренные символом %. Все остальные символы копируются в выходную строку как есть. Каждый спецификатор формата (т. е. символ % и следующие за ним команды) соответствует одному и только одному параметру, указанному после параметра $format. Если же нужно поместить в текст % как обычный символ, необходимо его удвоить:

```
echo sprintf("The percentage was %d%%",$percentage);
```

Каждый спецификатор формата включает максимум пять элементов (в порядке их следования после символа %).

1. Необязательный спецификатор размера поля, который указывает, сколько символов будет отведено под выводимую величину. В качестве символов-заполнителей (если значение имеет меньший размер, чем размер поля для его вывода) может использоваться пробел или 0, по умолчанию подставляется пробел. Можно задать любой другой символ-наполнитель, если указать его в строке форматирования, предварив апострофом '.

2. Опциональный спецификатор выравнивания, определяющий, будет результат выровнен по правому или по левому краю поля. По умолчанию производится выравнивание по правому краю, однако можно указать и левое выравнивание, задав символ «–» (минус).

3. Необязательное число, определяющее размер поля для вывода величины. Если результат не будет в поле помещаться, то он «вылезет» за края этого поля, но не будет усечен.

4. Необязательное число, предваренное точкой «.», предписывающее, сколько знаков после запятой будет в результирующей строке. Этот спецификатор учитывается только в том случае, если происходит вывод числа с плавающей точкой, в противном случае он игнорируется.

5. Обязательный спецификатор типа величины, которая будет помещена в выходную строку (табл. 5.2).

Таблица 5.2

Назначение обязательных спецификаторов

Спецификатор	Назначение
b	Очередной аргумент из списка выводится как двоичное целое число
c	Выводится символ с указанным в аргументе кодом
d	Целое число
f	Число с плавающей точкой
o	Восьмеричное целое число
s	Строка символов
x	Шестнадцатеричное целое число с маленькими буквами *a–z*
X	Шестнадцатеричное число с большими буквами *A–Z*

Вот как можно указать точность представления чисел с плавающей точкой:

```
<?PHP
    $money1 = 68.75;
    $money2 = 54.35;
    $money = $money1 + $money2;
    echo $money, "<br>";
    // echo $money выведет "123.1"...
    $formatted = sprintf ("%.2f", $money);
    // echo $formatted выведет "123.10"!
    echo $formatted;
    echo "<br><br>";
    echo $money, "<br>";
    // echo $money выведет "123.1"...
    $formatted = sprintf ("%d", $money);
    echo $formatted;
?>
```

Результат работы скрипта представлен на рис. 5.33.

Рис. 5.33. Пример форматирования денежных выражений с использованием функцией sprint()

Вот пример вывода целого числа, предваренного требуемым количеством нулей.

```
<?PHP

    $year = 2016;
    $month = 9;
    $day = 30;
    $isodate=sprintf("%04d-%02d-%02d",$year,$month,$day);
    echo $isodate;
    echo "<br>";
    $isodate=sprintf("%4d-%2d-%2d",$year,$month,$day);
    echo $isodate;

?>
```

Результат работы скрипта представлен на рис. 5.34.

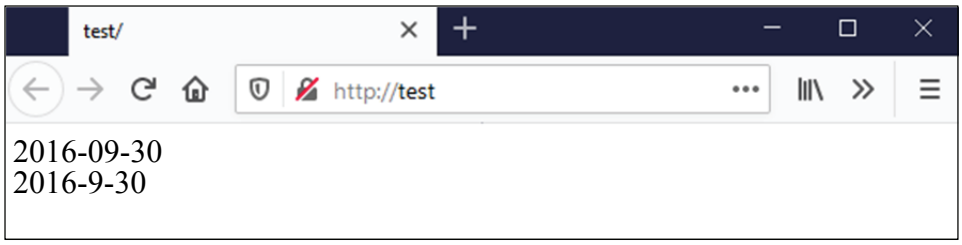

Рис. 5.34. Пример форматирования целого числа
с требуемым количеством нулей

Функция printf имеет следующий синтаксис:

```
int printf ( string $format [, mixed $args [, mixed $... ]] )
```

Выводит строку, отформатированную в соответствии с аргументом format (описание параметра format смотрите в описании функции sprintf()).

```
<?PHP
    printf("%.2f", 1.035);
    echo "<br>";
    printf("%.2f", round(1.035, 2));
?>
```

Результат работы скрипта представлен на рис. 5.35.

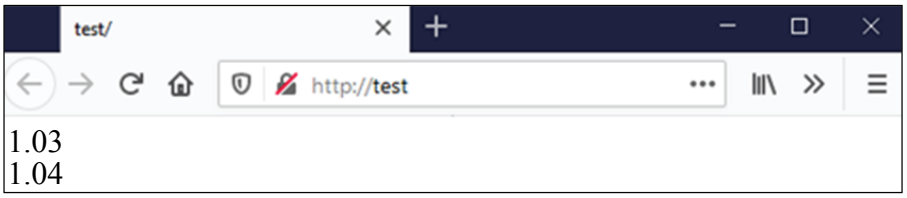

Рис. 5.35. Пример форматирования с использованием printf()

```
number_format(float $number, int $decimals, string
$dec_point=".", string $thousands_sep=",");
```

Эта функция форматирует число с плавающей точкой с разделением его на триады с указанной точностью. Она может быть вызвана с двумя или четырьмя аргументами, но не с тремя! Параметр $decimals задает, сколько цифр после запятой должно быть у чис-

ла в выходной строке. Параметр $dec_point представляет собой разделитель целой и дробной частей, а параметр $thousands_sep – разделитель триад в числе (если указать на его месте пустую строку, то триады не отделяются друг от друга).

В качестве примера приведем форматирование значений к французскому формату – обычно используются два знака после запятой (‘,’) и пробел (‘ ’) в качестве разделителя групп.

```
<?PHP
    $number=1234.56;
    $number_eng=number_format($number);
    print $number_eng;
    echo "<br>";
    $number_fr=number_format($number,2,',',' ');
    print $number_fr;
?>
```

Результат работы скрипта представлен на рис. 5.36.

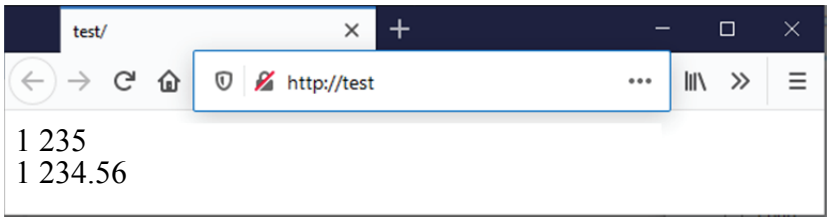

Рис. 5.36. Пример форматирования
с использованием функции number_format()

В PHP существует еще несколько функций для выполнения форматных преобразований, среди них – sscanf() и fscanf(), которые часто применяются в C. Однако в PHP их использование весьма ограничено: чаще всего для разбора строк оказывается гораздо выгоднее привлечь регулярные выражения или функцию explode().

Функция print_r выводит удобочитаемую информацию о переменной.

```
print_r ( mixed $expression [, bool $return = FALSE ] ) : mixed
```

Параметр expression определяет выражение для вывода на экран, а парметр return используется, если необходимо перехватить вывод *print_r()*. Если его значение равно TRUE, то print_r() вернет информацию вместо вывода в браузер.

Данная функция, равно как и *var_dump()*, может показывать защищенные и закрытые атрибуты объектов. Статические элементы класса не будут отображены.

Если в функцию передаются значения типов string, integer или float, будет напечатано само значение, если передается массив array, значения будут напечатаны в формате, показывающем ключи и элементы массива. Аналогичный формат вывода используется для объектов.

Если параметр return установлен в TRUE, данная функция вернет строку (string). В противном случае возвращаемое значение будет равно TRUE. Рассмотрим простейший пример вывода массива.

```
<?php
      $a = array ('a' => 'apple', 'b' => 'banana', 'c'
=> array ('x', 'y', 'z'));
      print_r ($a);
?>
```

Результаты выполнения кода представлена на рис. 5.37.

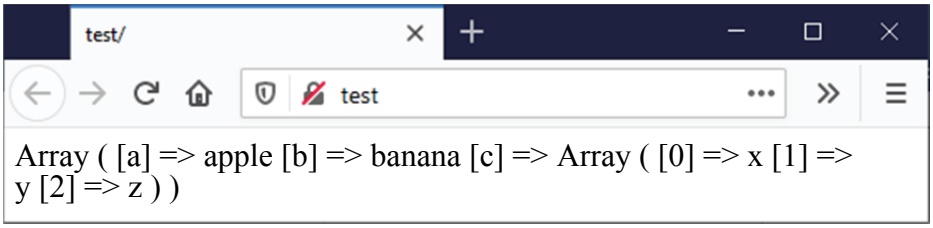

Рис. 5.37. Результаты использования print_r

Полный список функций для работы со строковыми переменными и их краткое описание представлены в прил. В.

Практическая часть

Пример 1

Рассмотрим следующую практическую задачу. Строковая переменная содержит последовательность из нескольких арабских цифр от 0 до 9. Если количество цифр меньше шести, должно выводиться сообщение об этом. Необходимо посчитать суммы пер-

вых трех цифр строки и трех последних. Если суммы будут равны, нужно вывести слово «Да», в противном случаи слово – «Нет». Код скрипта сохраним в файле index01.php.

```
<?php
  $str = '123456789';
  $slen = strlen($str); #определяем длину строки
  echo "Исходная строка $str";
  if( $slen < 6 )
  {
      echo "<br>Количество цифр меньше шести";
  }
  else
  {
      # берем первые три символа строки
      $str1 = substr($str,0,3);
      # берем последние три символа строки
      $str2 = substr($str,-3);
      $sum1 = $str1{0} + $str1{1} + $str1{2};
      $sum2 = $str2{0} + $str2{1} + $str2{2};
      echo "<br>Первые три символа строки \$str -
$str1. Их сумма равна $sum1";
      echo "<br>Последние три символа строки \$str -
$str2. Их сумма равна $sum2";
      if( $sum1==$sum2)
      {
        echo "<br>Да";
      }
      else {
        echo "<br>Нет";
      }
  }
?>
```

Результат выполнения скрипта представлен на рис. 5.38.

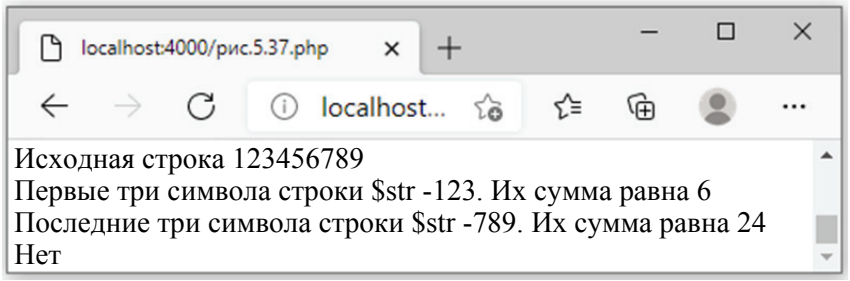

Рис. 5.38. Результат выполнения скрипта

Пример 2

Скрипт, представленный ниже, позволяет определить количество цифр и букв, входящих в MD5-хеш функции сроки символов. Для решения задачи с помощью функции str_replace() в исходной строке происходит замена цифр пустым символом. Скрипт сохраним в файле index02.php.

```
<?php
  $str = "Мы строили, строили и наконец построили.";
  $str_md5 = md5($str);
  echo "<br>Исходная строка: $str";
  echo "<br>MD5-хеш этой строки: $str_md5";

  # выделим из строки md5-хеш функции буквы
  # для этого создаем массив с цифрами
  $a_dig = array(1,2,3,4,5,6,7,8,9,0);
  $str_b = str_replace($a_dig,"",$str_md5);
  $slen_b = strlen($str_b);
  $slen_d = 32-$slen_b;
  echo "<br />Буквы, содержащиеся в MD5-хеш функции -
$str_b.";
  echo "<br />Количество букв в MD5-хеш функции - $slen_b";
  echo "<br />Количество цифр в MD5-хеш функции - $slen_d";
?>
```

Результат выполнения скрипта представлен на рис. 5.39.

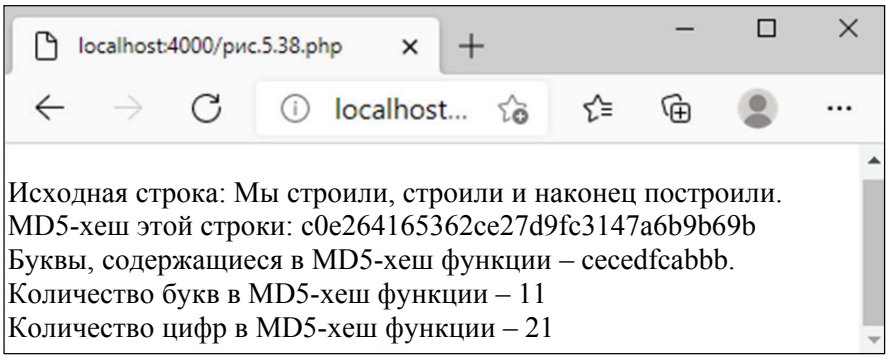

Рис. 5.39. Результат выполнения скрипта
по подсчету количества цифр и букв

Выполните модификацию скрипта таким образом, чтобы находилась сумма трех цифр начиная со второй, и сумма двух цифр, расположенных начиная с третьей от конца строки.

В целом отметим, что несмотря на достаточно большое число функций и разнообразие решаемых с их помощью типовых задач, как правило, на практике они применяются вместе с различными конструкциями (циклы, ветвления и т. д.), которые будут представлены в гл. 7.

Лабораторная работа № 5

Задания на лабораторную работу будут носить похожий на рассмотренные примеры характер и выдаваться индивидульно преподавателем. Напомним, что выполнение и демострация всех лабораторных работ предполагается в пределах структуры, представленной в лабораторной работе № 1.

ГЛАВА 6

ОПЕРАЦИИ НАД МАССИВАМИ

Массивы в PHP – это один из самых популярных на практике типов данных, который используется для структурирования и обработки большого количества данных. Рассмотрим некоторые часто используемые функции для работы с массивами.

6.1. Создание массива и извлечение информации из него

Функция list(). Предположим, есть массив, состоящий из трех элементов:

```
$names[0]="Александр";
$names[1]="Николай";
$names[2]="Яков";
```

Допустим, в какой-то момент нам нужно передать значения всех трех элементов массива, соответственно трем переменным: **$alex**, **$nick**, **$yakov**. Это можно сделать следующим образом:

```
$alex = $names[0];
$nick = $names[1];
$yakov = $names[2];
```

Если массив большой, то такой способ присвоения его элементов массива переменным не очень удобен. Есть более рациональный подход – использование функции list().

```
    list ($alex, $nick, $yakov) = $names;
```

Если нужны только «Николай» и «Яков», то можно сделать так:

```
list (, $nick, $yakov) = $names;
```

Приведем простейший пример.

```
<?PHP
    $names[0]="Александр";
    $names[1]="Николай";
    $names[2]="Яков";
    var_dump ($names);
    list($alex, $nick, $yakov)=$names;
    var_dump ($alex);
    var_dump ($nick);
    var_dump ($yakov);
?>
```

Результаты работы данного скрипта представлены на рис. 6.1.

Приведем пример, позволяющий записывать в переменные лишь часть значений массива (результаты представлены на рис. 6.2).

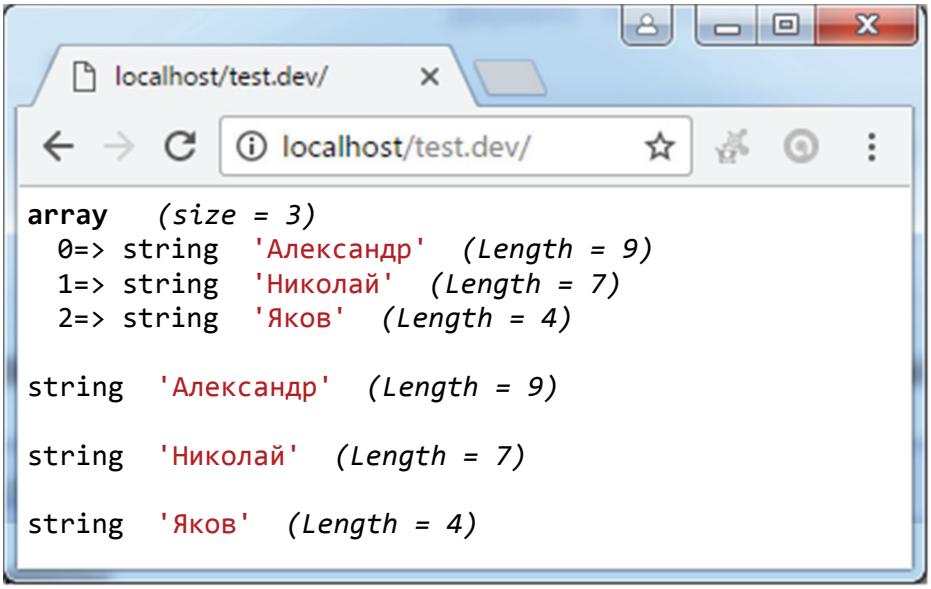

Рис. 6.1. Пример использования функции list()

```
<?PHP
    $names[0]="Александр";
    $names[1]="Николай";
    $names[2]="Яков";
    var_dump ($names);
    list($alex,, $yakov)=$names;
    var_dump ($alex);
    var_dump ($yakov);
?>
```

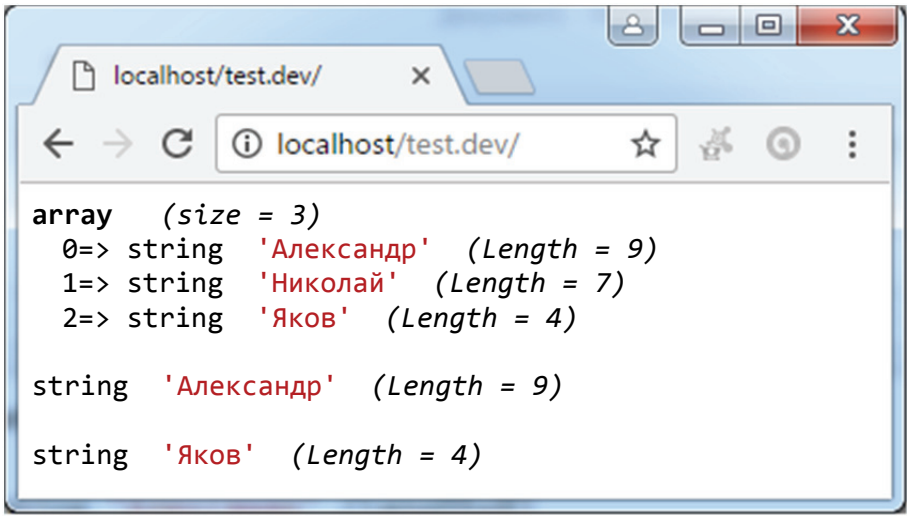

Рис. 6.2. Пример использования функции list()
для части элементов массива

Функция array(). Функция Array() используется специально для создания массивов. При этом она позволяет создавать пустые массивы. Вот методы использования функции Array():

```
<?PHP
    $arr = array(); // Создает пустой массив
    $arr2 = array("Иванов","Петров","Сидоров"); // Со-
здает список с тремя элементами. Индексы начинаются с
нуля
    $arr3 = array("Иванов"=>"Иван", "Петров"=>"Петр",
"Сидоров"=>"Сидор"); // Создает ассоциативный массив с
тремя элементами
    $arr4 = array("name"=>"Иванов", "age"=>"24",
"email"=>"ivanov@mail.ru"); // Создает многомерный ас-
социативный массив
    $arr4 = array("name"=>"Петров", "age"=>"34",
"email"=>"petrov@mail.ru");
    $arr4 = array("name"=>"Сидоров", "age"=>"47",
"email"=>"sidorov@mail.ru");
    var_dump ($arr);
    var_dump ($arr2);
    var_dump ($arr3);
    var_dump ($arr4);
?>
```

Результаты скрипта представлены на рис. 6.3.

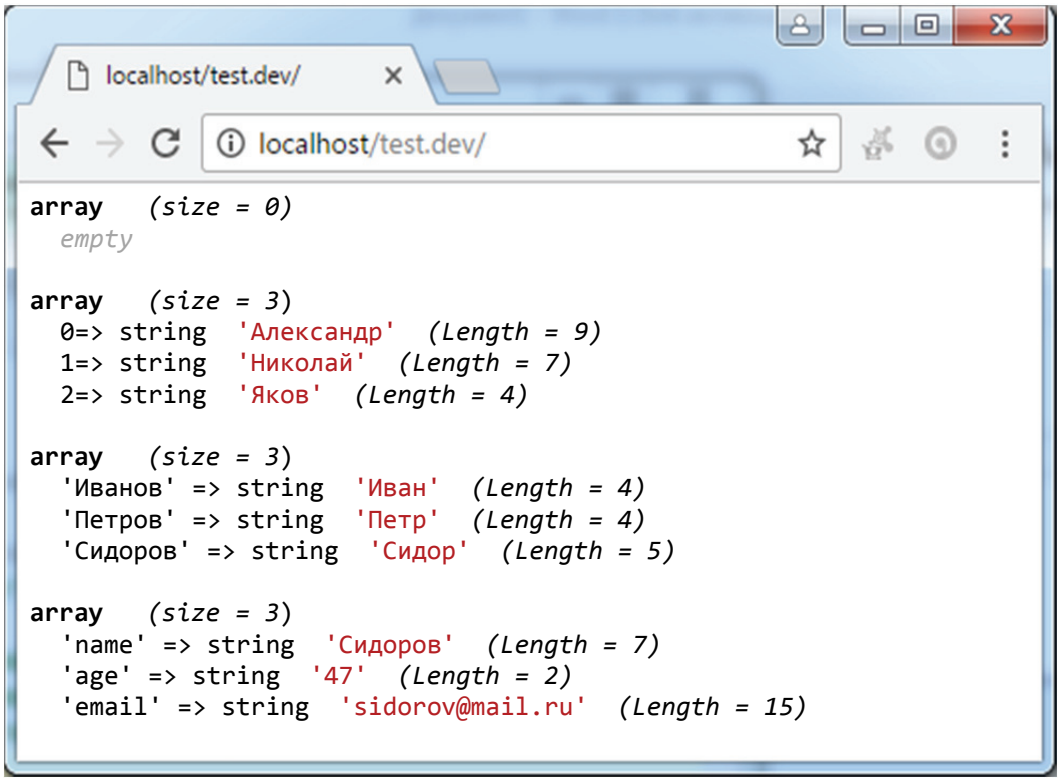

Рис. 6.3. Использование функции array() для создания массива

Разберите полученные результаты самостоятельно.

6.2. Функции для работы с массивами

6.2.1. Сортировка массивов. Начнем с самого простого – сортировки массивов. В PHP для этого существует очень много функций. С их помощью можно сортировать ассоциативные массивы и списки в порядке возрастания или убывания, а также в том порядке, в каком вам необходимо посредством пользовательской функции сортировки.

Функции asort() и arsort() используются для сортировки массива по значениям. Функция asort() сортирует массив, указанный в ее параметре так, чтобы его значения шли в алфавитном (если это строки) или возрастающем (для чисел) порядке. При этом сохраняются связи между ключами и соответствующими им значениями, т. е. некоторые пары ключ=>значение просто «всплывают» наверх, а некоторые – наоборот, «опускаются».

```
<?PHP
    $A=array("a"=>"Zero","b"=>"Weapon","c"=>"Alpha","d"
=>"Processor");
    var_dump ($A);
    asort($A);
    var_dump($A);
?>
```

Результат выполнения скрипта представлен на рис. 6.4.

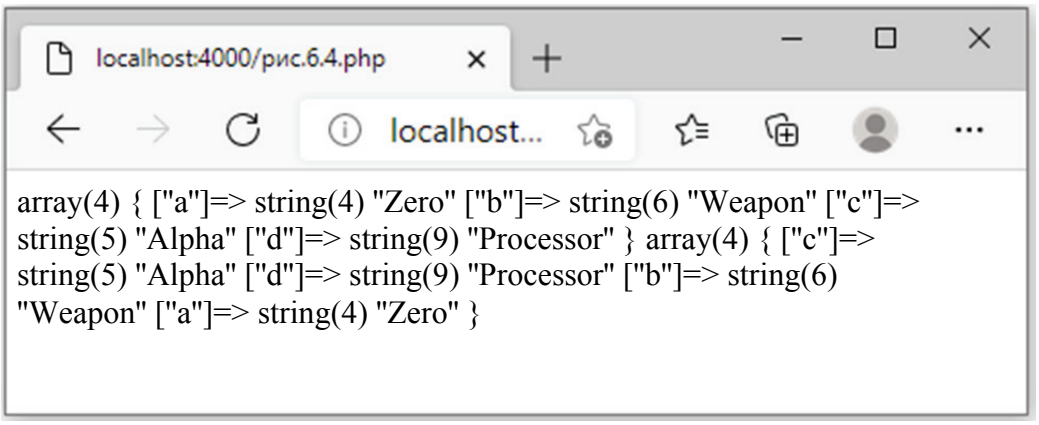

Рис. 6.4. Пример использования функции asort()

Функция arsort() выполняет то же самое, за одним исключением: она упорядочивает массив не по возрастанию, а по убыванию.

Функции ksort() и krsort() используются для сортировки по ключам. Функция ksort() практически идентична функции asort(), с тем различием, что сортировка осуществляется не по значениям, а именно по ключам (в порядке возрастания). Например:

```
<?PHP
    $A=array("b"=>"Zero","f"=>"Weapon","d"=>"Alpha","c"
=>"Processor");
    var_dump ($A);
    ksort($A);
    var_dump($A);
?>
```

Результат выполнения скрипта представлен на рис. 6.5.

Функция для сортировки по ключам в обратном порядке называется krsort() и применяется точно в таком же контексте, что и ksort().

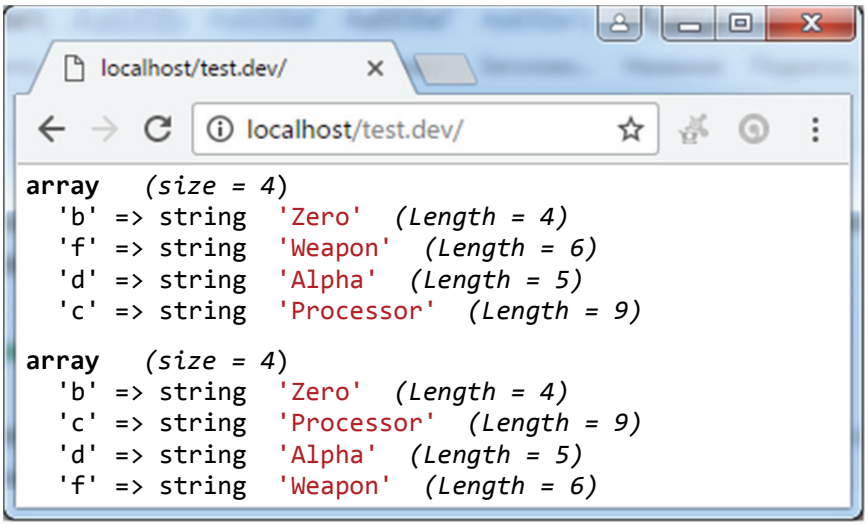

Рис. 6.5. Пример использования функции ksort()

Функции uksort() и uasort(). Довольно часто нам приходится сортировать что-то по более сложному критерию, чем просто по алфавиту. В таком случае можно использовать функцию *uksort()*. Например, пусть в $Files хранится список имен файлов и подкаталогов в текущем каталоге. Возможно, мы захотим вывести этот список не только в лексикографическом порядке, но также, чтобы все каталоги предшествовали файлам. В этом случае стоит воспользоваться функцией uksort(), написав предварительно функцию сравнения с двумя параметрами, как того требует uksort(). Связи между ключами и значениями сохраняются функцией uksort(), т. е. опять же, некоторые пары просто «всплывают» наверх, а другие – «оседают».

Функция uasort() очень похожа на uksort(), с той лишь разницей, что пользовательской функции сортировки будут передаваться не ключи, а очередные значения из массива. При этом также сохраняются связи в парах ключ=>значение.

Функция array_multisort() может быть использована для сортировки сразу нескольких массивов или одного многомерного массива в соответствии с одной или несколькими размерностями.

При этом ассоциативные (string) ключи будут сохранены, но числовые ключи переиндексированы.

```
array_multisort ( array &$array1 [, mixed $array1_sort_order
= SORT_ASC [, mixed $array1_sort_flags =
SORT_REGULAR [, mixed $... ]]] ) : bool
```

Рассмотрим параметры функции:

− *array1* – сортируемый массив (array);

− *array1_sort_order* – порядок для сортировки вышеуказанного аргумента типа array (SORT_ASC для сортировки по возрастанию, или SORT_DESC для сортировки по убыванию). Этот аргумент может меняться местами с *array1_sort_flags* или вообще быть пропущенным. В таком случае подразумевается значение SORT_ASC;

− *array1_sort_flags* – настройки сортировки для вышеуказанного аргумента array. При этом используются следующие флаги:

• SORT_REGULAR – обычное сравнение элементов (без изменения типов);

• SORT_NUMERIC – сравнение элементов как чисел;

• SORT_STRING – сравнение элементов как строк;

• SORT_LOCALE_STRING – сравнение элементов как строк, учитывая текущую локаль. Используется локаль, которую можно менять с помощью функции setlocale();

• SORT_NATURAL – сравнение элементов как строк с использованием алгоритма «natural order», как в функции natsort();

• SORT_FLAG_CASE – может быть объединен (бинарное ИЛИ) с SORT_STRING или SORT_NATURAL для сортировки без учета регистра.

Отметим, что этот аргумент может меняться местами с *array1_sort_order* или вообще быть пропущенным. В этом случае подразумевается значение SORT_REGULAR.

Рассмотрим пример сортировки многомерного массива.

```
<?PHP
    $ar = array(
           array("10", 11, 100, 100, "a"),
           array(   1,  2, "2",   3,   1)
          );
    array_multisort($ar[0], SORT_ASC, SORT_STRING,
                    $ar[1], SORT_NUMERIC, SORT_DESC);
    foreach ($ar as $key1 => $value1) {
        echo "[$key1] строка массива".':<br>';
        foreach ($value1 as $key2 => $value2) {
            echo $key2.'=>';
            var_dump($value2);
            echo "<br>";
        }
        echo "<br>";
    };
?>
```

Результаты работы скрипта представлены на рис. 6.6.

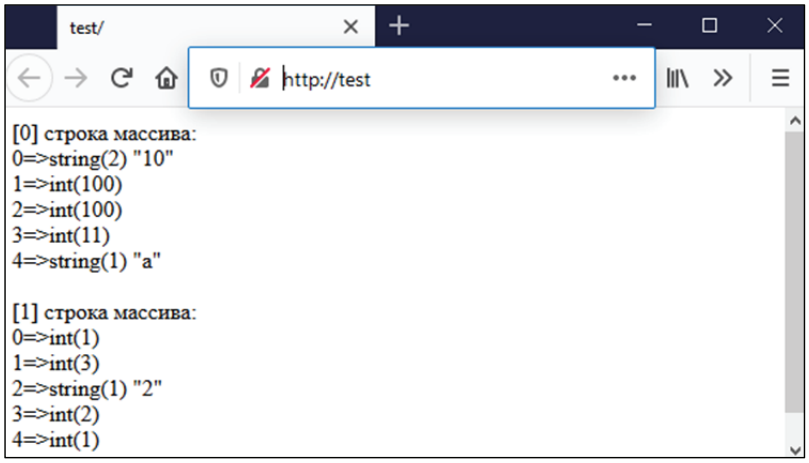

Рис. 6.6. Пример использования функции ***array_multisort()***

Функция array_reverse(). Необходимо отметить, что это не функция сортировки в прямом смысле – она возвращает массив, элементы которого следуют в обратном порядке относительно массива, переданного в параметре. При этом связи между ключами и значениями, конечно, не теряются. Например, вместо того чтобы ранжировать массив в обратном порядке при помощи arsort(), мы можем отсортировать его в прямом порядке, а затем перевернуть:

```
<?PHP
    $A=array("b"=>"Zero","f"=>"Weapon","d"=>"Alpha","c"
=>"Processor");
    var_dump ($A);
    $B=array_reverse($A);
    var_dump($B);
?>
```

Результаты работы скрипта представлены на рис. 6.7.

Можно выполнять подобные операции и с ассоциативным массивом.

```
<?PHP
    $A=array("b"=>"Zero","f"=>"Weapon","d"=>"Alpha","c"
=>"Processor");
    var_dump ($A);
    asort($A);
    var_dump($A);
    $B=array_reverse($A);
    var_dump($B);
?>
```

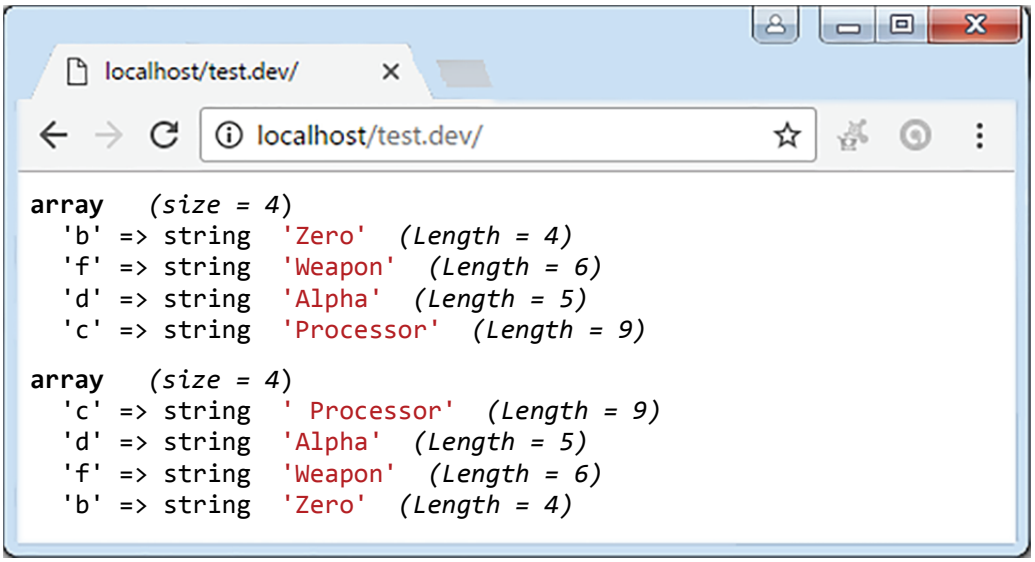

Рис. 6.7. Пример использования функции ***array_reverse()***

Результаты работы скрипта представлены на рис. 6.8.

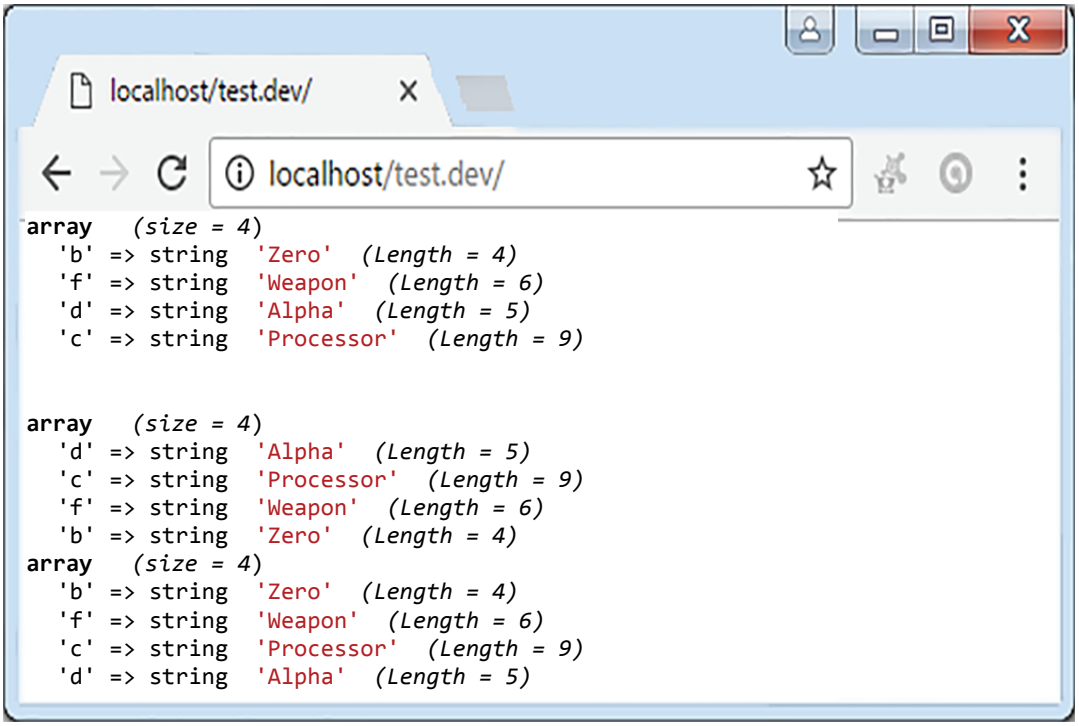

Рис. 6.8. Пример использования функции ***array_reverse()*** с ассоциативным массивом

Однако надо отметить, что указанная последовательность работает дольше, чем один единственный вызов arsort().

Функции sort() и rsort() предназначены в первую очередь для сортировки списков.

Функция sort() сортирует список (разумеется, по значениям) в порядке возрастания, а rsort() – в порядке убывания. Пример для функции sort():

```
<?PHP
    $A=array("40", "20", "10", "30");
    var_dump($A);
    sort($A);
    var_dump ($A);
?>
```

Результаты работы скрипта представлены на рис. 6.9.

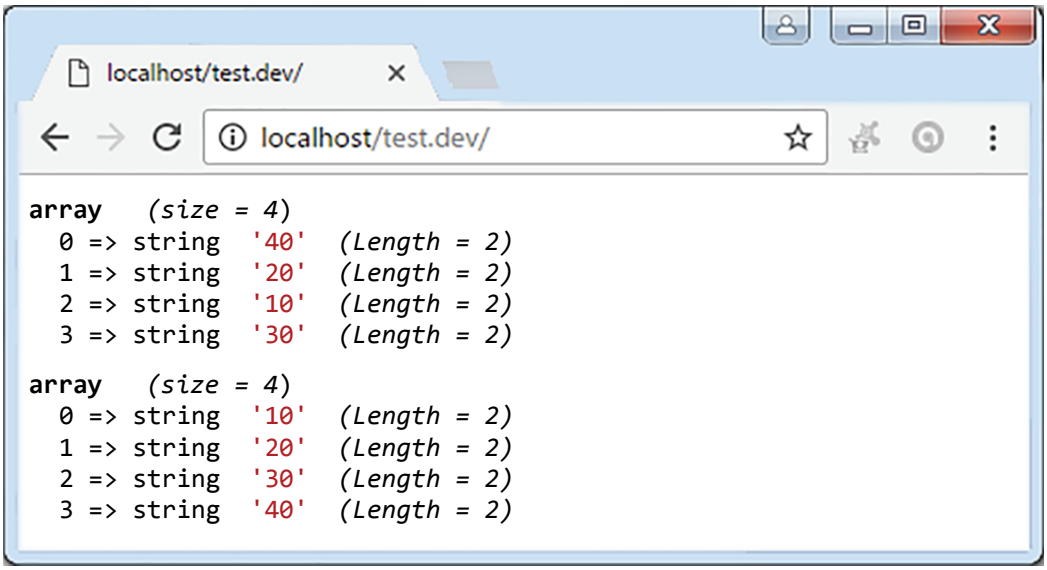

Рис. 6.9. Пример использования функции **sort()** для списка

Функция shuffle() позволяет перемешивать список, переданный ей первым параметром так, чтобы его значения распределялись случайным образом. Обратите внимание, что, во-первых, изменяется сам массив, а во-вторых, ассоциативные массивы воспринимаются как списки.

```
<?PHP
    $A=array("40", "20", "10", "80", "50", "90","30","70");
    var_dump($A);
    sort($A);
```

```
    var_dump ($A);
    shuffle($A);
    foreach($A as $V) echo $V." ";
    echo "<br>";
    shuffle($A);
    foreach($A as $V) echo $V." ";
    echo "<br>";
    shuffle($A);
    foreach($A as $V) echo $V." ";
?>
```

Результаты работы скрипта представлены на рис. 6.10.

После обновления страницы получим другой случайный порядок следования чисел (рис. 6.11).

Выполнив этот фрагмент несколько раз, может возникнуть ситуация, в которой очередность следования чисел не изменится.

Это свойство обусловлено тем, что функция **shuffle()** использует стандартный генератор случайных чисел, который перед работой необходимо инициализировать при помощи вызова *srand*().

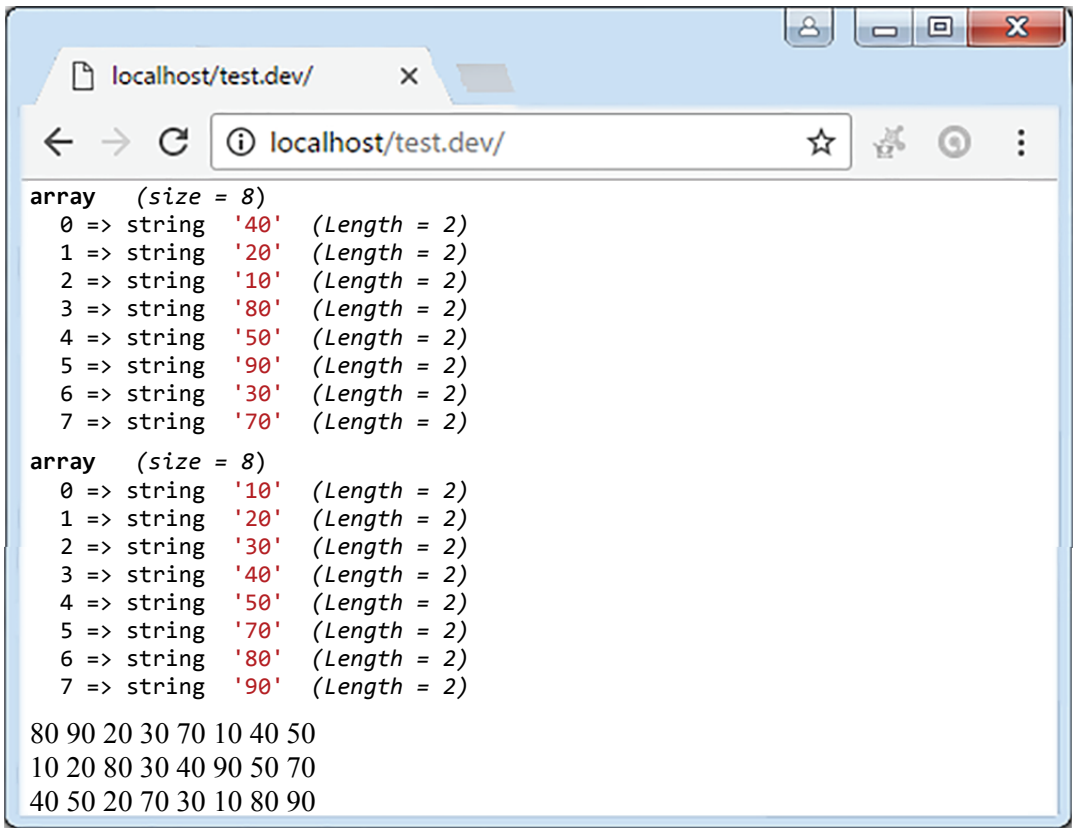

Рис. 6.10. Пример использования функции **shuffle()**

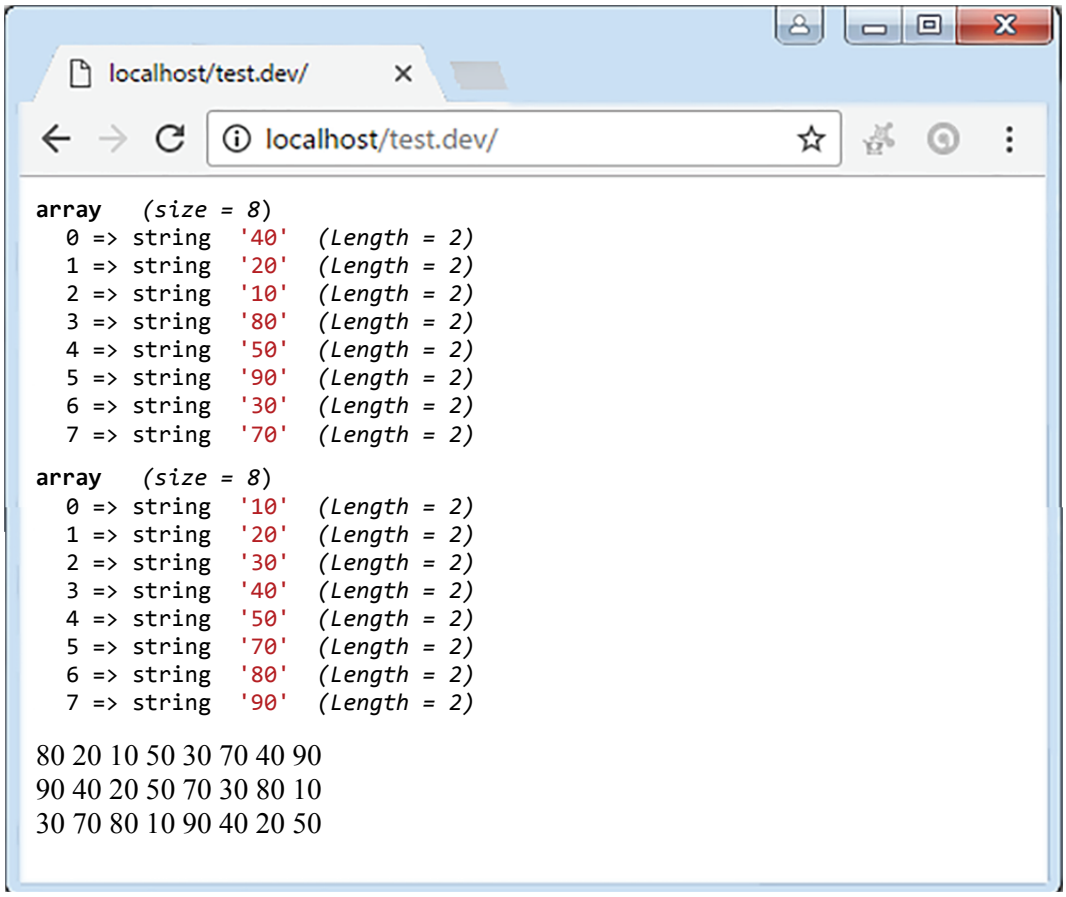

Рис. 6.11. Пример повторного использования функции **shuffle()**

6.2.2. Операции с ключами и значениями массива. Данная группа функций направлена на выполнение каких-либо дейстий с отдельными элементами массива:

1) array_flip(array $arr)

Функция **array_flip**() проходит по массиву и меняет местами его ключи и значения. Исходный массив $arr не изменяется, а результирующий массив просто возвращается. При этом если в массиве присутствовали несколько элементов с одинаковыми значениями, учитываться будет только последний из них.

```
<?PHP
    $A=array("40", "20", "10", "80", "50", "90","30","70");
    var_dump($A);
    $A=array_flip($A);
    var_dump($A);
?>
```

Результаты работы скрипта представлены на рис. 6.12.

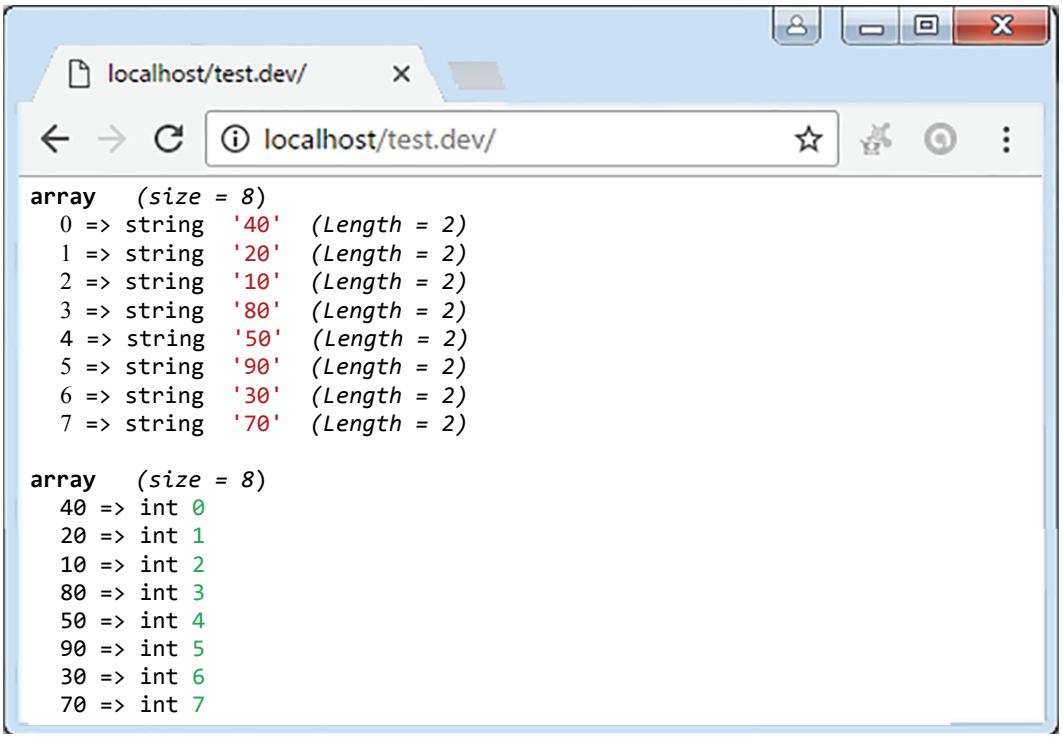

Рис. 6.12. Пример использования функции **array_flip()**

2) array_keys(array $arr [,mixed $SearchVal])

Функция **array_keys()** возвращает список, содержащий все ключи массива $arr. Рассмотрим пример.

```
<?PHP

    $A=array("a"=>"40", "b"=>"20","c"=> "10",
"b"=>"80", "e"=>"50", "f"=>"90","d"=>"30","h"=>"70");
    var_dump($A);
    $B=array_keys($A);
    var_dump($B);

?>
```

Результаты работы скрипта представлены на рис. 6.13.

Если задан необязательный параметр $SearchVal, то она вернет только те ключи, которым соответствуют значения $SearchVal. Фактически, эта функция с заданным вторым параметром является обратной по отношению к оператору [ ] – извлечению значения по его ключу.

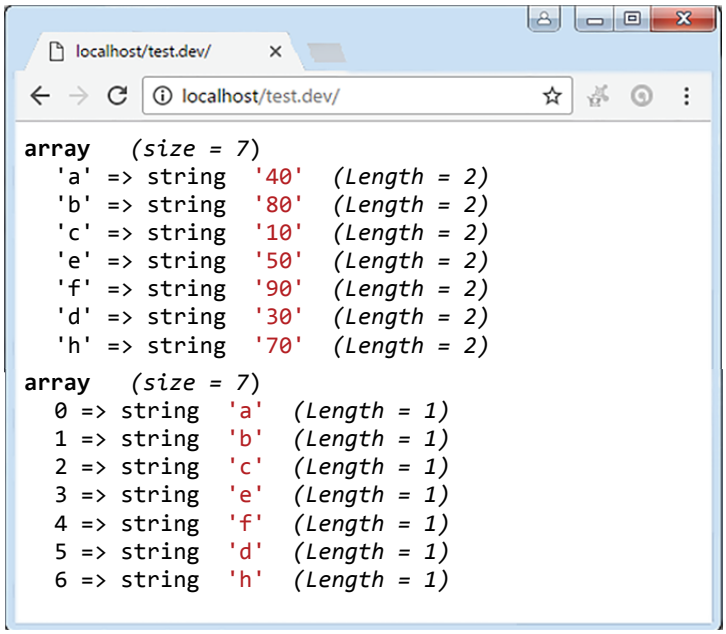

Рис. 6.13. Пример использования функции **array_keys()**

```
<?PHP
    $A=array("a"=>"40", "b"=>"20","c"=> "10",
"b"=>"80", "e"=>"50", "f"=>"90","d"=>"30","h"=>"70");
    var_dump($A);
    $B=array_keys($A,"80");
    var_dump($B);
?>
```

Результаты работы скрипта представлены на рис. 6.14.

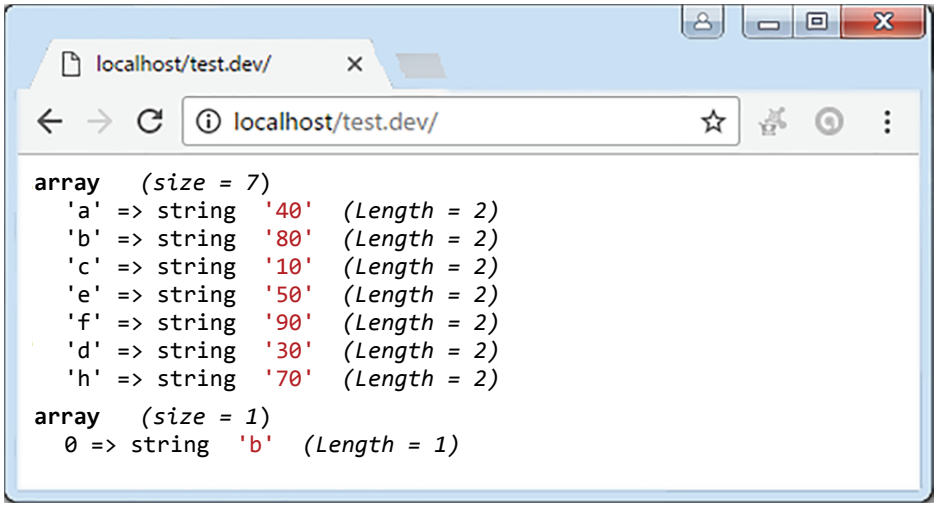

Рис. 6.14. Пример использования функции **array_keys()**
с использованием дополнительного параметра

3) array_values(array $arr)

Функция array_values() возвращает список всех значений в ассоциативном массиве **$arr**. Она также заново индексирует возвращаемый массив. Очевидно, такое действие бесполезно для списков, но иногда оправдано для хешей.

```
<?PHP
    $A=array("a"=>"40", "b"=>"20","c"=> "10",
"b"=>"80", "e"=>"50", "f"=>"90","d"=>"30","h"=>"70");
    var_dump($A);
    $B=array_values($A);
    var_dump($B);
?>
```

Результаты работы скрипта представлены на рис. 6.15.

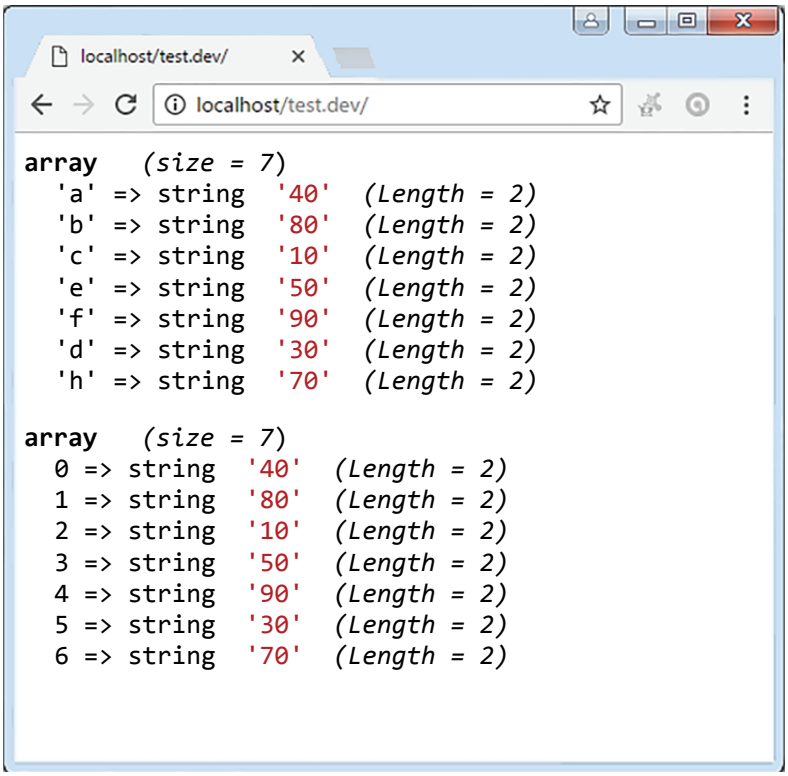

Рис. 6.15. Пример использования функции **array_values()**

4) in_array(mixed $val, array $arr)

Функция **in_array()** возвращает **true**, если элемент со значением **$val** присутствует в массиве **$arr**. Впрочем, если часто приходится проделывать эту операцию, то необходимо решить, воз-

можно лучше будет воспользоваться ассоциативным массивом и хранить данные в его ключах, а не в значениях. На этом можно сильно выиграть в быстродействии.

```
<?PHP
    $A=array("a"=>"40", "b"=>"20","c"=> "10",
"b"=>"80", "e"=>"50", "f"=>"90","d"=>"30","h"=>"70");
    var_dump($A);
    $B=in_array(50,$A);
    var_dump($B);
    $B=in_array(55,$A);
    var_dump($B);
?>
```

Результаты работы скрипта представлены на рис. 6.16.

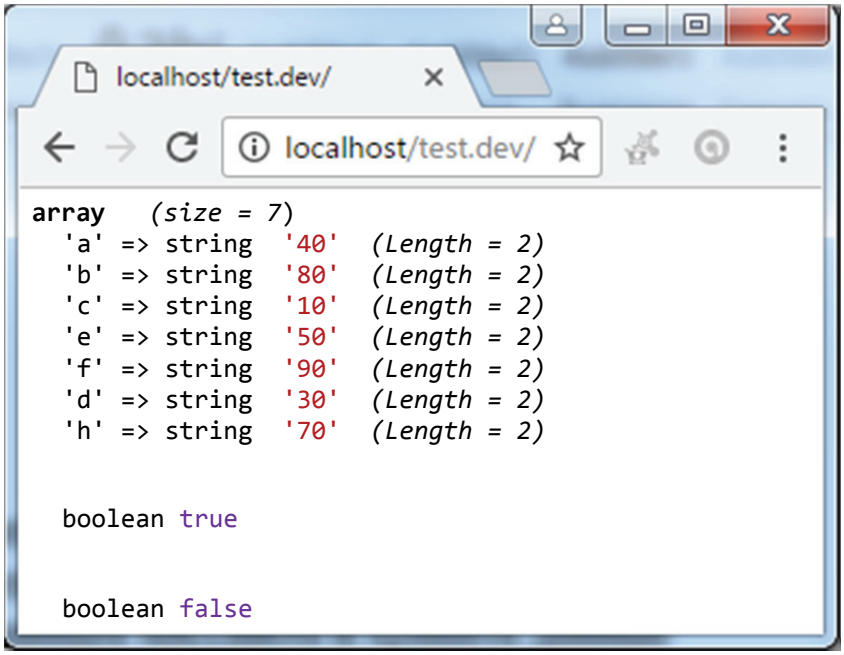

Рис. 6.16. Пример использования функции **in_array()**

5) array_count_values(list $List)

Функция array_count_values() подсчитывает, сколько раз каждое значение встречается в списке $List, и возвращает ассоциативный массив с ключами – элементами списка и значениями – количеством повторов этих элементов. Иными словами, функция array_count_values() подсчитывает частоту появления значений в списке $List.

```
<?PHP
    $A=array("a"=>"40", "b"=>"50","c"=> "70",
"d"=>"80", "e"=>"50", "f"=>"90","d"=>"50","h"=>"70");
    var_dump($A);
    $B=array_count_values($A);
    var_dump($B);
?>
```

Результаты работы скрипта представлены на рис. 6.17.

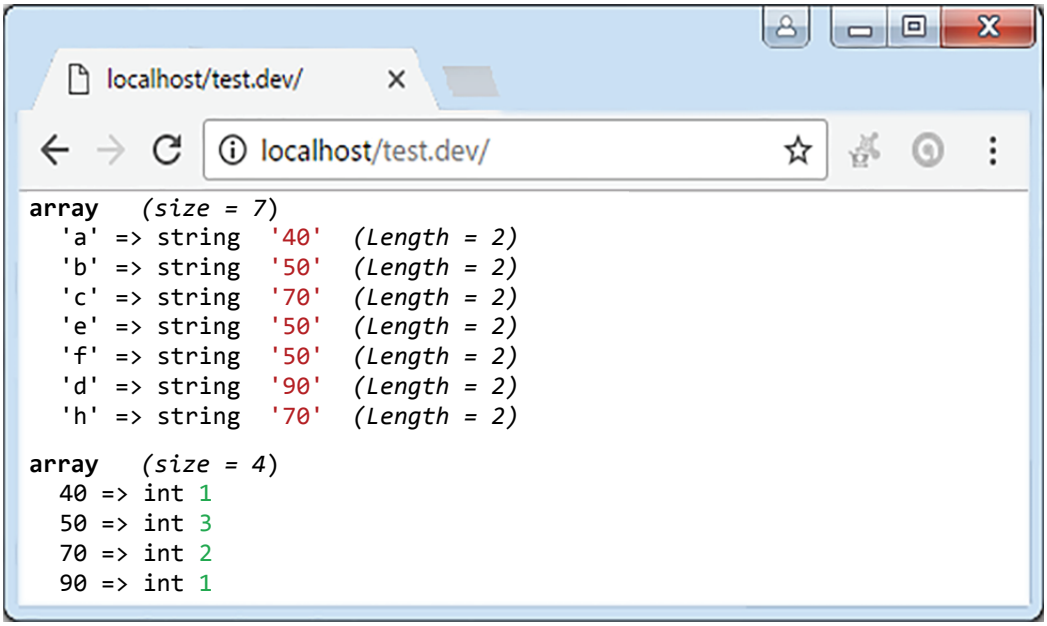

Рис. 6.17. Пример использования функции **count_values()**

В примере приведен вариант применения функции count_values.

6.2.3. Конкатенация массивов. Слияние (конкатенация) массивов – это операция создания массива, состоящего из элементов нескольких других массивов. Слияние массивов – очень опасная операция, поскольку результат слияния подчиняется своей логике, забыв о которой можно потерять данные. Слияние массивов реализуется при помощи оператора «+» или с помощью функции array_merge(). Слияние списков может осуществляться только с помощью функции array_merge().

Предположим, имеются два списка.

```
$A = array("1"=>"первый", "2"=>"Второй");
$B = array("1"=>"Третий", "2"=>"Четвертый");
```

Теперь объединим данные двух списков в один.

```
$C = $A + $B;
```

Оператор «+» для массивов не коммутативен. Это означает, что **$A + $B** не равно **$B + $A**. В результате рассмотренного примера мы получим список **$C** следующего вида:

```
"1"=>"Первый", "2"=>"Второй"
```

А в результате **$B + $A** мы получим такой список:

```
"1"=>"Третий", "2"=>"Четвертый"
```

Связано это с тем, что при слиянии списков с одинаковыми индексами в результирующем списке остается элемент первого массива, причем на том же месте. Аналогично дело обстоит и с двумерными массивами.

```
<?PHP
    $A[]= array(0=>1, 1=>2);
    $A[] = array(0=>3, 1=>4);
    $B[] = array(0=>5, 1=>6);
    $B[] = array(0=>7, 1=>8);
    $B[]=array(0=>9, 1=>10);
    echo "Массив A  +  Массив B","<br>";
    $C = $A + $B;
    var_dump($C);
    echo "Массив B  +  Массив A","<br>";
    $C=$B+$A;
    var_dump($C);
?>
```

Результаты работы скрипта представлены на рис. 6.18.

В таком случае необходимо использовать функцию array_merge(), которая призвана устранить все недостатки, присущие оператору «+» для слияния массивов. А именно, она объединяет массивы, перечисленные в ее аргументах, в один большой массив и возвращает результат. Если в массивах встречаются одинаковые ключи, в результат помещается пара *ключ=>значение* из того массива, который расположен правее в списке аргументов. Однако

это не затрагивает числовые ключи: элементы с такими ключами помещаются в конец результирующего массива в любом случае. Таким образом, с помощью array_merge() можно избавиться от всех недостатков оператора «+» для массивов. Вот пример, объединяющий два списка в один:

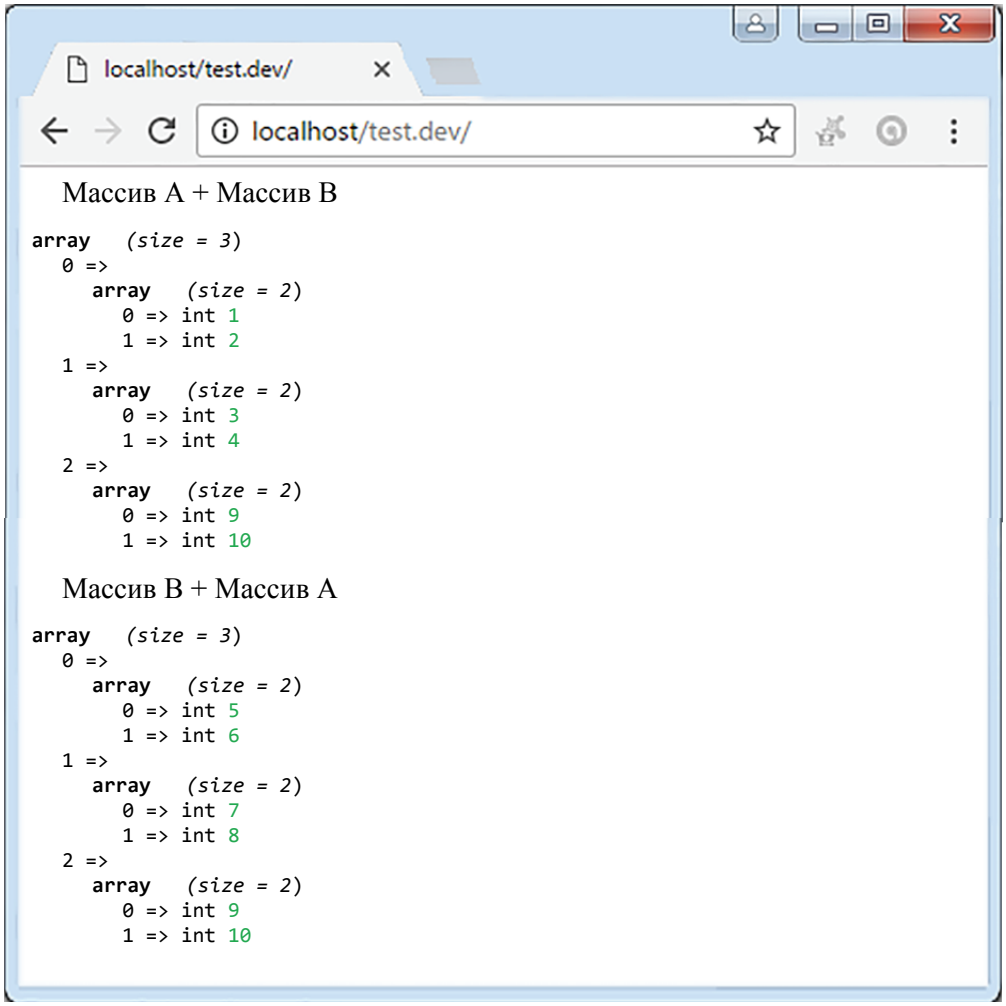

Рис. 6.18. Конкатенация массивов

```
<?PHP
    $A= array("1"=>"первый", "2"=>"второй");
    $B = array("1"=>"третий",
"2"=>"четвертый","3"=>"пятый");
    var_dump($A);
    var_dump($B);

    echo "Массив A  +  Массив B","<br>";
    $C = $A + $B;
```

```
    var_dump($C);

    echo "Массив B  +  Массив A","<br>";
    $C=$B+$A;
    var_dump($C);

    echo 'array_merge($A,$B)',"<br>";
    $C=array_merge($A,$B);

    var_dump($C);
?>
```

Результаты работы скрипта представлены на рис. 6.19.

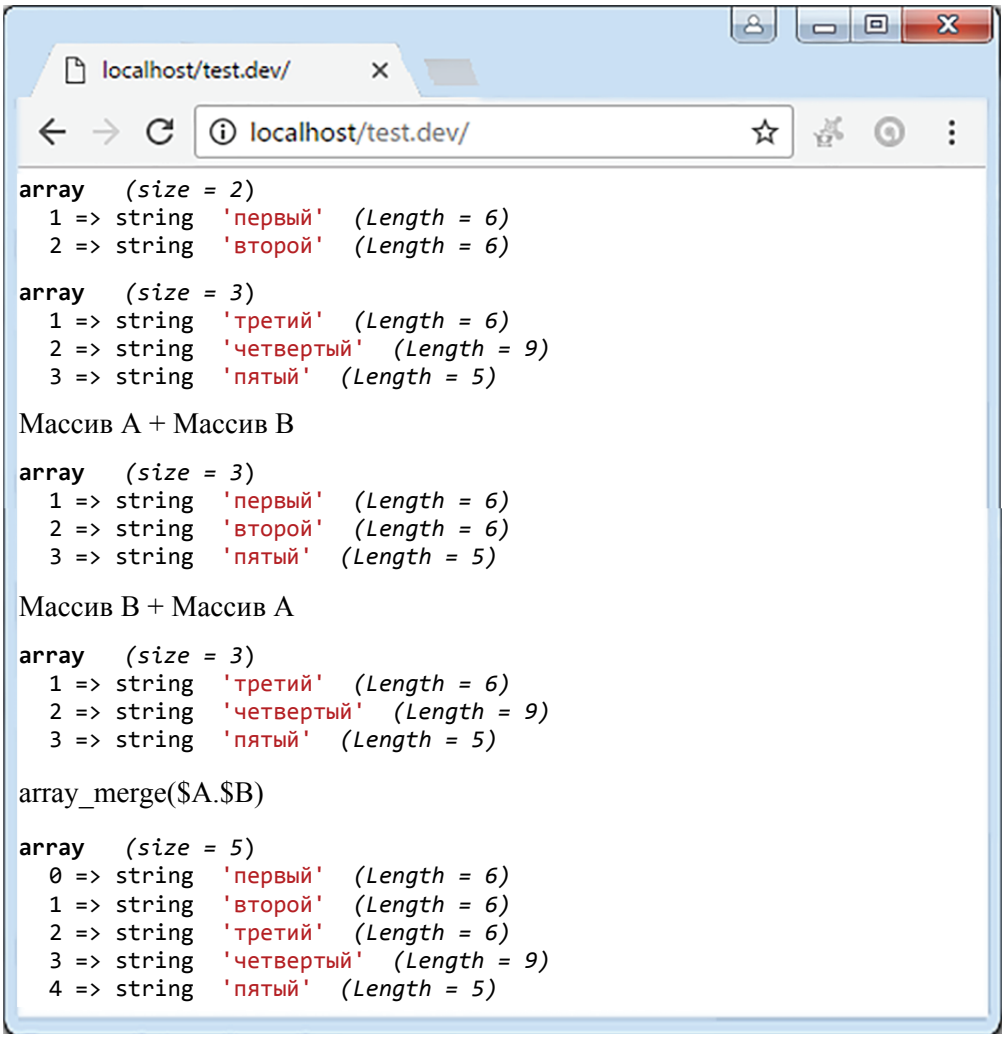

Рис. 6.19. Конкатенация массивов с использованием функции array_merge()

Мы можем наблюдать объединение двух массивов в один.

6.2.4. Получение части массива. Для получения части массива можно использовать функцию **array_slice()**.

```
array_slice(array $Arr, int $offset [, int $len])
```

Эта функция возвращает часть ассоциативного массива начиная с пары **ключ=>значения** со смещением (номером) **$offset** от начала и длиной **$len** (если последний параметр не задан – до конца массива). Параметры **$offset** и **$len** задаются по точно таким же правилам, как и аналогичные параметры в функции **substr()**: они могут быть отрицательными (в этом случае отсчет осуществляется от конца массива) и т. д. Вот несколько примеров.

```
<?PHP
    $input = array ("a", "b", "c", "d", "e");
    $output = array_slice ($input, 2); // "c", "d", "e"
    var_dump($output);
    $output = array_slice ($input, -2, 2); // "d", "e"
    var_dump($output);
    $output = array_slice ($input, 0, 3); // "a", "b", "c"
    var_dump($output);
?>
```

Результаты работы скрипта представлены на рис. 6.20.

Рис. 6.20. Использование функции array_slice()

Применение функций позволяет сократить объем кода и получить максимальную производительность при исполнении скрипта.

6.2.5. Вставка и удаление элементов массивов. Наиболее простым и уже знакомым является использование следующих операторов, которые отвечают за вставку и удаление элементов: например, оператор [] (пустые квадратные скобки) добавляет элемент в конец массива, присваивая ему числовой ключ, а оператор Unset() вместе с извлечением по ключу может быть использован как для удаления отдельного элемента массива, так и всего массива целиком.

```
<?PHP
    $arr = array(5 => 1, 12 => 2);
    var_dump($arr);
    $arr[] = 56;    // В этом месте скрипта это эквива-
лентно $arr[13] = 56;
    var_dump($arr);
    $arr["x"] = 42; // Это добавляет к массиву новый
элемент с ключом "x"
    var_dump($arr);
    unset($arr[5]); // Это удаляет элемент из массива
    var_dump($arr);
    unset($arr);    // Это удаляет массив полностью
    var_dump($arr);
?>
```

Результаты работы скрипта представлены на рис. 6.21.

Язык PHP также поддерживает и многие другие функции, которые иногда бывает удобно использовать.

```
array_push(alist &$Arr, mixed $var1 [, mixed $var2, …])
```

Эта функция добавляет к списку **$Arr** элементы **$var1**, **$var2** и т. д. Она присваивает им числовые индексы – точно так же, как это происходит для стандарных []. Если нужно добавить всего один элемент, наверное, проще и будет воспользоваться этим оператором, в противном случае рекомендуется использовать функцию array_push().

```
<?PHP
    $A = array ("a", "b", "c", "d", "e");
    var_dump($A);
    array_push($A,"f","g");
    var_dump($A);
?>
```

Результаты работы скрипта представлены на рис. 6.22.

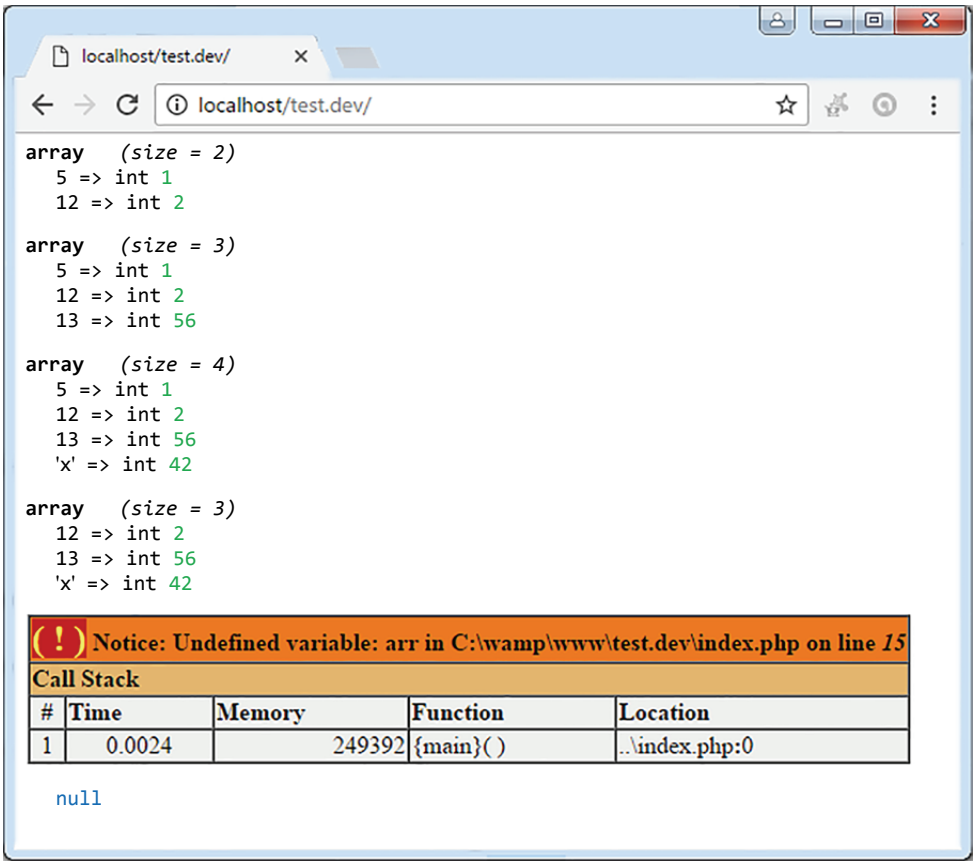

Рис. 6.21. Добавление и удаление элементов массива

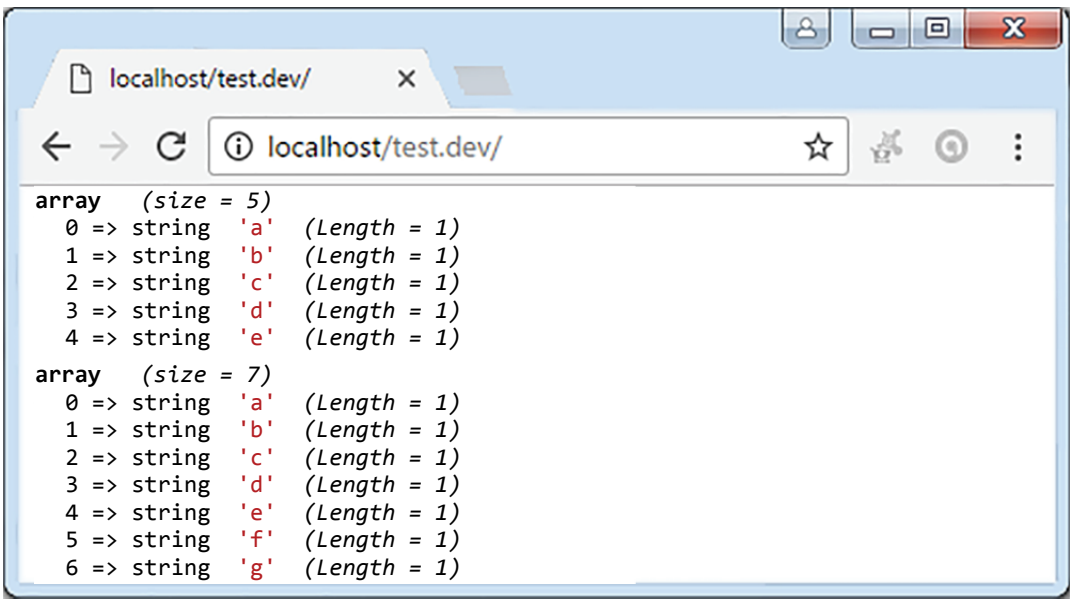

Рис. 6.22. Использование функции array_push()

Обратите внимание, что функция **array_push**() воспринимает массив как стек и добавляет элементы всегда в его конец. Она возвращает новое число элементов в массиве.

Функция array_pop() является противоположностью array_push(), снимает элемент с «вершины» стека, т. е. берет последний элемент списка, и возвращает его, удалив после этого его из **$Arr**.

```
array_pop(list &$Arr)
```

С помощью данной функции мы можем строить конструкции, напоминающие стек. Если список **$Arr** был пуст, функция возвращает пустую строку.

```
<?PHP
    $A = array ("a", "b", "c", "d", "e");
    var_dump($A);
    array_push($A,"f","g");
    var_dump($A);
    $B=array_pop($A);
    var_dump($A);
    echo "удаленное значение =$B";
?>
```

Результаты работы скрипта представлены на рис. 6.23.

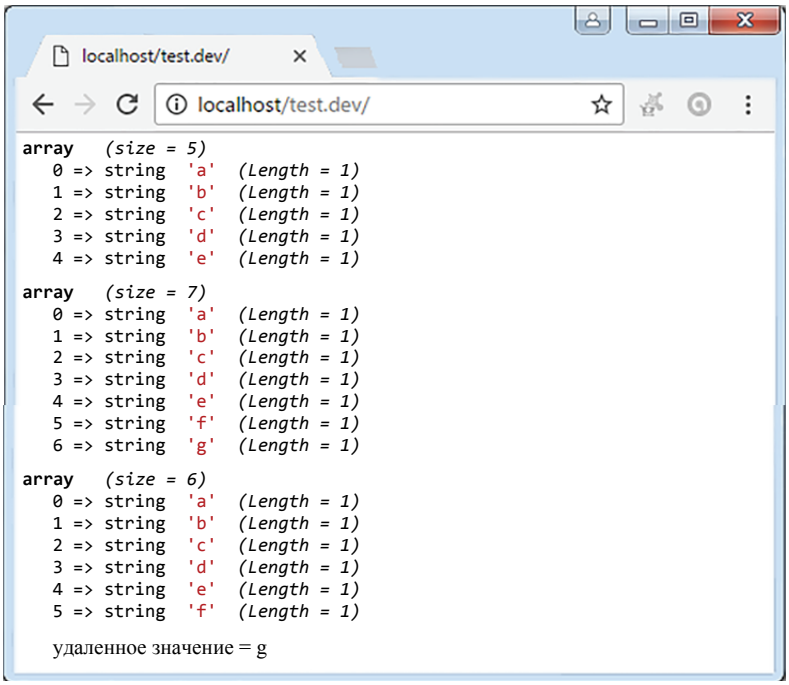

Рис. 6.23. Использование функции array_pop()

Функция array_unshift очень похожа на array_push(), но добавляет перечисленные элементы не в конец, а в начало массива.

```
array_unshift(list &$Arr, mixed $var1 [, mixed $var2, …])
```

При этом порядок следования **$var1**, **$var2** и т. д. остается тем же, т. е. элементы как бы «вдвигаются» в список слева. Новым элементам списка, как обычно, назначаются числовые индексы начиная с **0**; при этом все ключи старых элементов массива, которые также были числовыми, изменяются (чаще всего они увеличиваются на число вставляемых значений). Функция возвращает новый размер массива. Вот пример ее применения:

```
<?PHP
    $A = array ("a", "b", "c", "d", "e");
    var_dump($A);
    array_unshift($A,"f","g");
    var_dump($A);
?>
```

Результаты работы скрипта представлены на рис. 6.24.

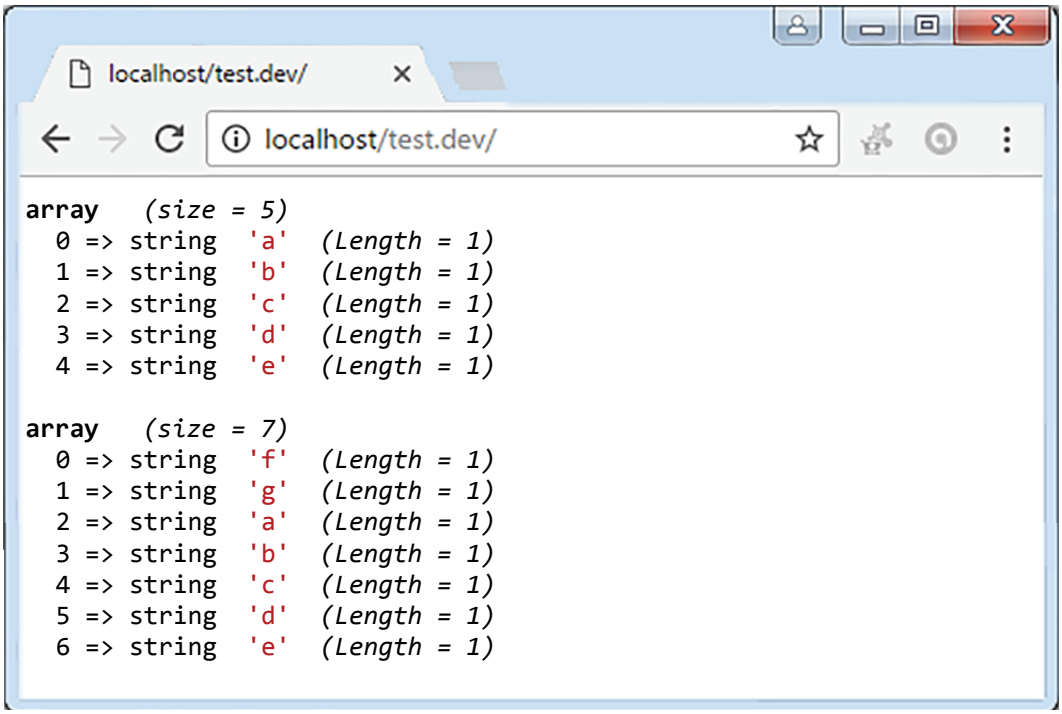

Рис. 6.24. Использование функции array_unshift()

Функция array_shift извлекает первый элемент массива **$Arr** и возвращает его.

```
    array_shift(list &$Arr)
```

Она сильно напоминает array_pop(), но только получает начальный, а не конечный элемент, а также производит довольно сильную «встряску» всего массива: ведь при извлечении первого элемента приходится корректировать все числовые индексы у всех оставшихся элементов.

```
<?PHP
    $A = array ("a", "b", "c", "d", "e");
    var_dump($A);
    array_shift($A);
    var_dump($A);
?>
```

Результаты работы скрипта представлены на рис. 6.25.

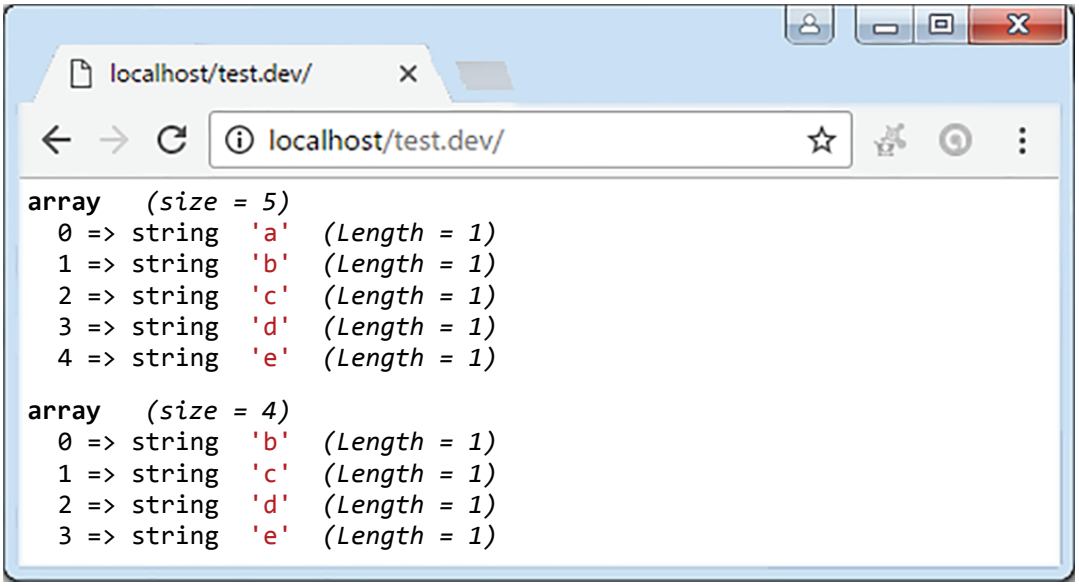

Рис. 6.25. Использование функции array_shift()

Функция **array_unique()** возвращает массив, составленный из всех уникальных значений массива **$Arr** вместе с их ключами.

```
array_unique(array $Arr)
```

В результирующий массив помещаются первые встретившиеся пары ключ=>значение.

```
<?PHP
    $A = array ("a", "b", "b", "d", "e",  "c", "c", "e");
    var_dump($A);
    $B=array_unique($A);
    var_dump($B);
?>
```

Результаты работы скрипта представлены на рис. 6.26.

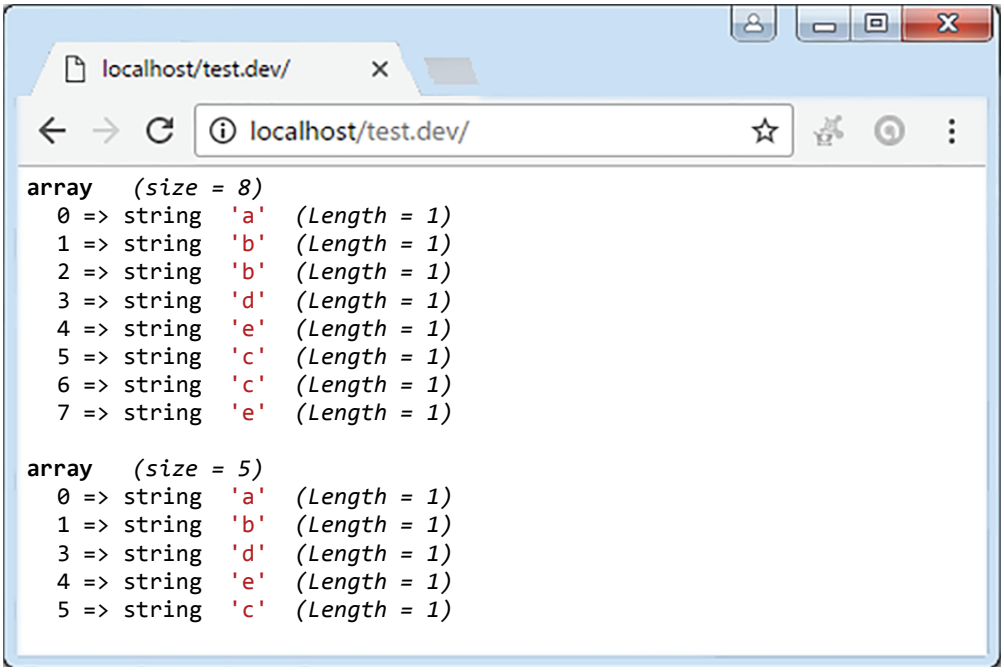

Рис. 6.26. Использование функции array_unique()

Функция array_splice, также как и array_slice(), возвращает подмассив **$Arr** начиная с индекса **$offset** максимальной длины **$len**, но вместе с тем она делает и другое полезное действие.

```
array_splice(array &$Arr, int $offset [, int $len] [, int $Repl])
```

При этом она заменяет только что указанные элементы на то, что находится в массиве **$Repl** (или просто удаляет, если **$Repl** не указан). Параметры **$offset** и **$len** задаются так же, как и в функции **substr()**: они могут быть и отрицательными, в этом случае отсчет начинается от конца массива. Вот некоторые примеры.

```
<?PHP
    $input=array("red", "green", "blue", "yellow");
    array_splice($input,2);
    var_dump($input);

    $input=array("red", "green", "blue", "yellow");
    array_splice($input, 1, 2, array("black", "grey"));
    var_dump($input);
    $input=array("red", "green", "blue", "yellow");
    array_splice($input, 1, 1, array("black", "grey"));
    var_dump($input);
    $input=array("red", "green", "blue", "yellow");
    array_splice($input, 2, 0, array("black", "grey"));
    var_dump($input);
    $input=array("red", "green", "blue", "yellow");
    array_splice($input, 1, 2, "black");
    var_dump($input);
?>
```

Результаты работы скрипта представлены на рис. 6.27.

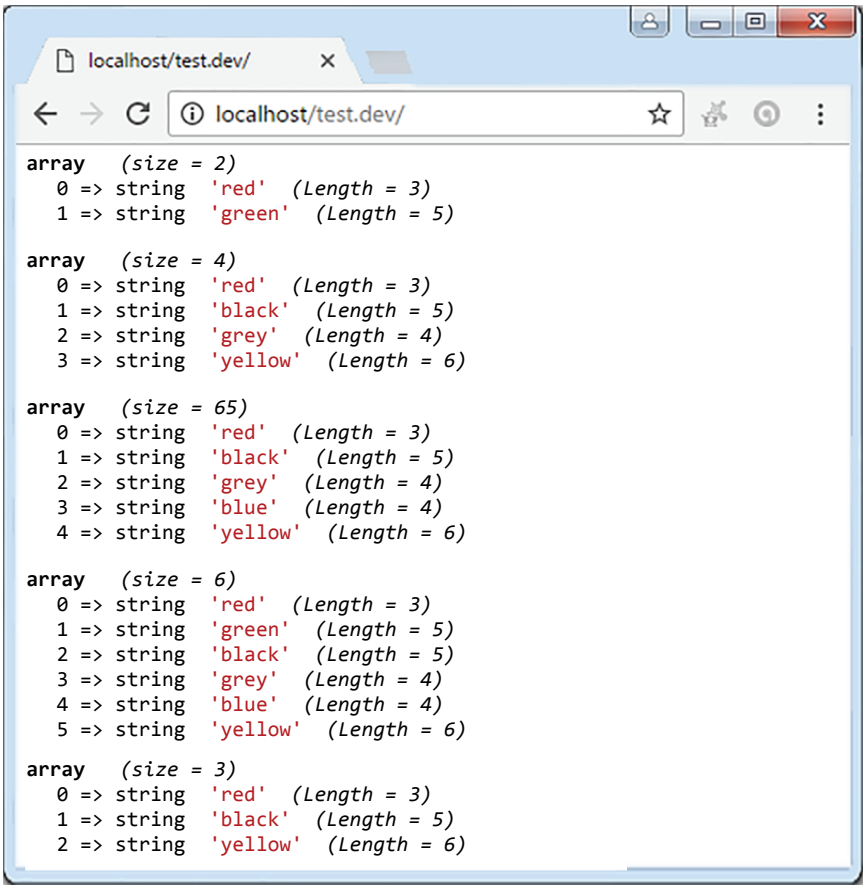

Рис. 6.27. Использование функции array_splice()

6.2.6. Преобразования переменных в массив и обратно. Для подобных операций могут быть использованы несколько функций.

Функция **compact()** упаковывает в массив переменные из текущего контекста (глобального или контекста функции), заданные своими именами в **$vn1**, **$vn2** и т. д.

```
compact(mixed $vn1 [, mixed $vn2, …])
```

При этом в массиве образуются пары с ключами, равными содержимому **$vnN**, и значениями соответствующих переменных. Вот пример использования этой функции:

```
<?PHP
    $a="Test string";
    $b="Some text";
    $A=compact("a","b");
    var_dump($A);
    //теперь $A===array("a"=>"Test string", "b"=>"Some
text")
?>
```

Результаты работы скрипта представлены на рис. 6.28.

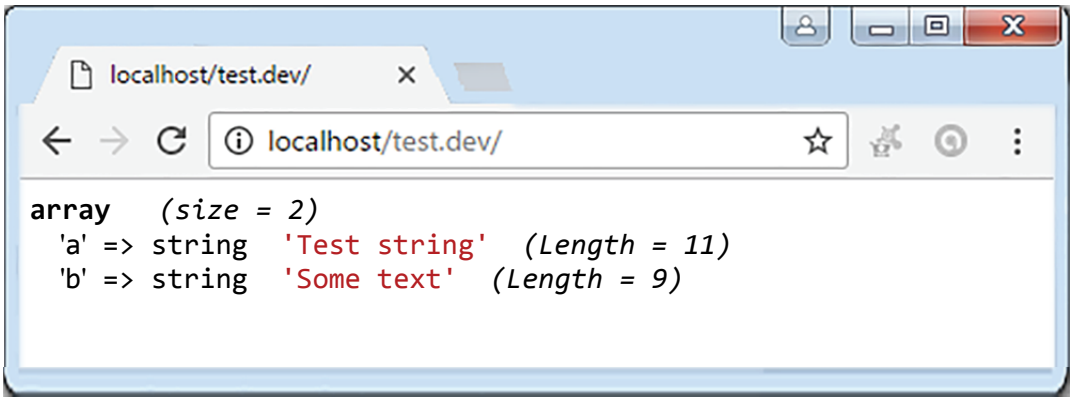

Рис. 6.28. Использование функции compact()

Отметим, что параметры функции обозначены как mixed. Это связано с тем, что они могут быть не только строками, но и списками строк. В этом случае функция последовательно перебирает все элементы данного списка и упаковывает те переменные из текущего контекста, имена которых она встретила. Более того

эти списки могут в свою очередь также содержать списки строк и т. д. Правда, последнее используется сравнительно редко, но все же вот пример:

```
<?PHP
    $a="Test";
    $b="Text";
    $c="CCC";
    $d="DDD";
    $Lst=array("b",array("c","d"));
    var_dump($Lst);
    $A=compact("a",$Lst);
    var_dump($A);
    //   теперь   $A===array("a"=>"Test",   "b"=>"Text",
"c"=>"CCC", "d"=>"DDD")
?>
```

Результаты работы скрипта представлены на рис. 6.29.

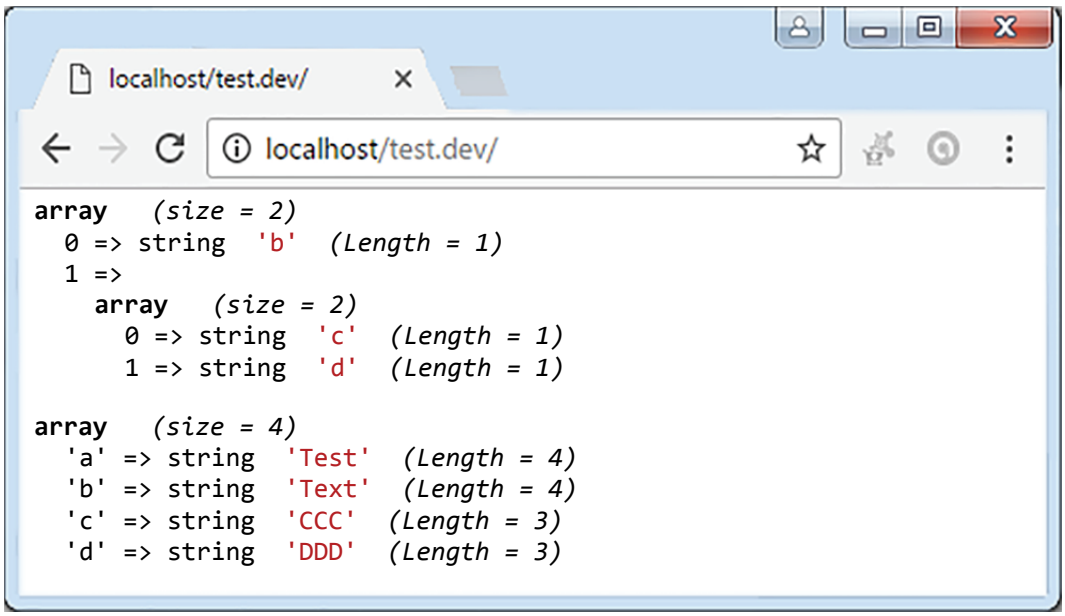

Рис. 6.29. Использование функции compact()
с получением многоуровневого массива

Функция extract() производит действия, прямо противоположные compact().

```
extract(array $Arr [, int $type] [, string $prefix])
```

Таким образом, она получает в параметрах массив **$Arr** и превращает каждую его пару ключ=>значение в переменную текущего контекста.

Для создания списка, представляющего собой диапазона чисел, используется функция range().

```
range(int $low, int $high)
```

Эта функция очень простая. Она создает список, заполненный целыми числами от **$low** до **$high** включительно.

```
<?PHP
    $low=10;
    $high=15;
    $A=range($low,$high);
    var_dump($A);
?>
```

Результаты работы скрипта представлены на рис. 6.30.

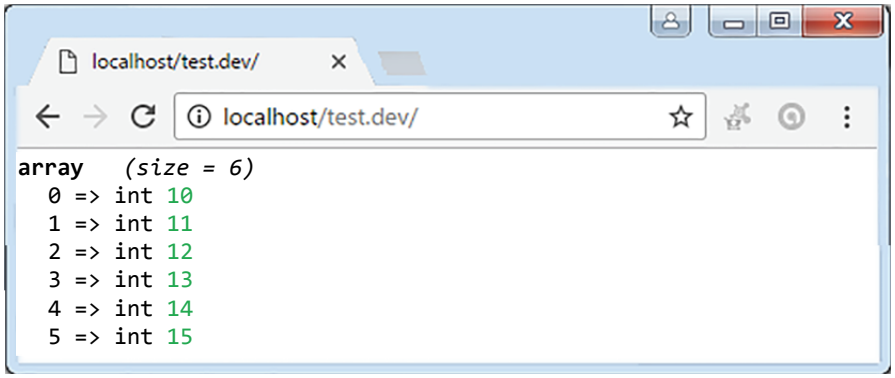

Рис. 6.30. Использование функции range()

Для подсчета элементов массива предназначена функция count(). Пример использования функции **count()**:

```
<?PHP
    $low=10;
    $high=15;
    $A=range($low,$high);
    var_dump($A);
    echo "количество элементов в массиве =".count($A);
?>
```

Результаты работы скрипта представлены на рис. 6.31.

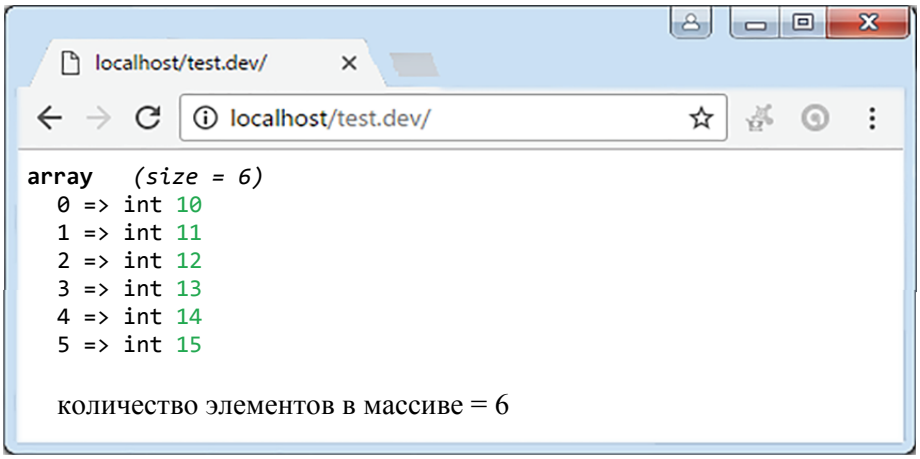

Рис. 6.31. Использование функции count()

Полный список функций для работы с массивами и их краткое описание представлены в прил. Г.

6.3. Функции для сравнения массивов

Если нужно проверить, какой элемент совпадает в том или ином массиве, или наоборот, не совпадает, то можно использовать функцию array_diff.

```
<?php
     $arr[1] = "PHP";
     $arr[2] = "HTML";
     $arr[3] = "JS" ;
     $arr[4] = "CSS";

     $arr2[1] = "PHP";
     $arr2[2] = "NODE.JS";
     $arr2[3] = "PYTHON";
     $arr2[4] = "CSS";

     $diff = array_diff($arr, $arr2);
     print_r ($diff);
?>
```

Результаты работы скрипта представлены на рис. 6.32.

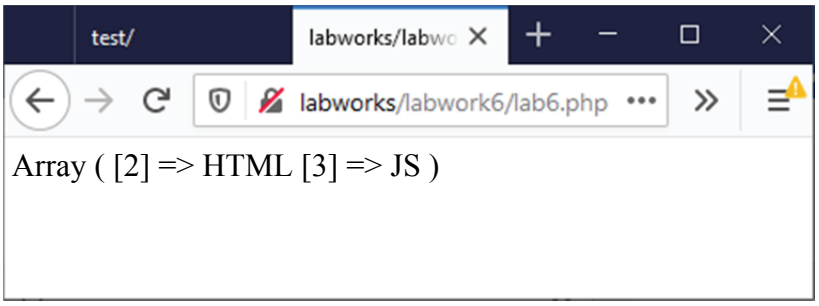

Рис. 6.32. Использование функции array_diff()

Если нужно, чтобы сравнивался еще и строковый индекс массивов, то тут уже следует использовать функцию array_diff_assoc.

```
<?php
     $arr[1] = "PHP";
     $arr[2] = "HTML";
     $arr[3] = "JS" ;
     $arr[4] = "CSS";

     $arr2[1] = "PHP";
     $arr2[2] = "NODE.JS";
     $arr2[3] = "PYTHON";
     $arr2[5] = "CSS";

     $diff = array_diff_assoc($arr, $arr2);
     print_r ($diff);
?>
```

Результаты работы скрипта представлены на рис. 6.33.

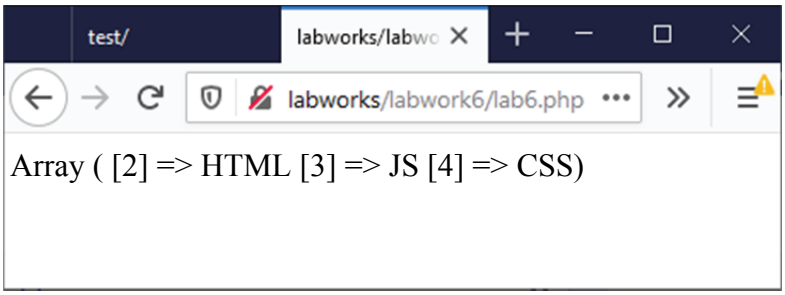

Рис. 6.33. Использование функции array_diff_assoc()

И наоборот, если требуется найти общие элементы массивов, то поможет функция array_intersect.

```
<?php
     $arr[1] = "PHP";
     $arr[2] = "HTML";
     $arr[3] = "JS" ;
     $arr[4] = "CSS";

     $arr2[1] = "PHP";
     $arr2[2] = "NODE.JS";
     $arr2[3] = "PYTHON";
     $arr2[5] = "CSS";

     $diff = array_intersect($arr, $arr2);
     print_r ($diff);
?>
```

Результаты работы скрипта представлены на рис. 6.34.

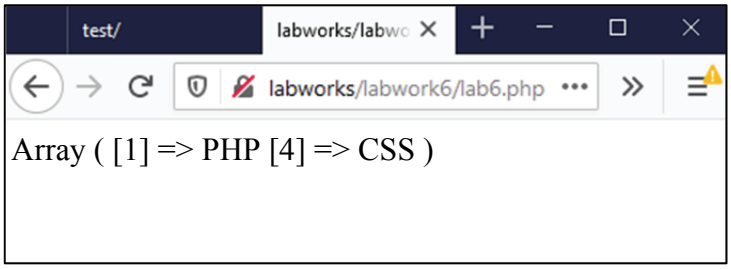

Рис. 6.34. Использование функции array_intersect()

Если при нахождении общих элементов массивов нужно учесть и индекс, то по аналогии array_diff_assoc можно использовать функцию array_intersect_assoc.

6.4. Особенности приведения других типов данных в массив

Для любого из типов: integer, float, string, boolean и resource, если вы преобразуете значение в массив, то получите массив с одним элементом (с индексом 0), являющимся скалярным значением, с которого вы начали.

Если вы преобразуете в массив объект (object), вы получите в качестве элементов массива свойства (переменные-члены) этого объекта. Ключами будут имена переменных-членов.

Если вы преобразуете в массив значение NULL, вы получите пустой массив.

Практическая часть

Приведем некоторые полезные практические примеры по работе с массивами.

Пример 1

Рассмотрим различные варианты ввода и вывода массивов.

```
<?PHP

   $a = array( 'color' => 'red',
               'taste' => 'sweet',
               'shape' => 'round',
               'name'  => 'apple',
               4        // ключом будет 0
             );

   // будет получен тот же результат
   $a['color'] = 'red';
   $a['taste'] = 'sweet';
   $a['shape'] = 'round';
   $a['name']  = 'apple';
   $a[]        = 4;        // ключом будет 0
?>
```

```
<?php
    $b[] = 'a';
    $b[] = 'b';
    $b[] = 'c';
    // создаст масив array(0 => 'a' , 1 => 'b' , 2 => 'c'),
    // или просто array('a', 'b', 'c')
?>
```

Еще один практический пример – введем двумерный массив.

```
<?php
    $A=array(0=>array('red', 'green'),
               1=>array('blue', 'yellow'),
               2=>array('grey','black', 'white')
              );
    print_r($A);
    echo '<br><br>';
    var_dump ($A[0][1]);
?>
```

Результат выполнения кода представлен на рис. 6.35.

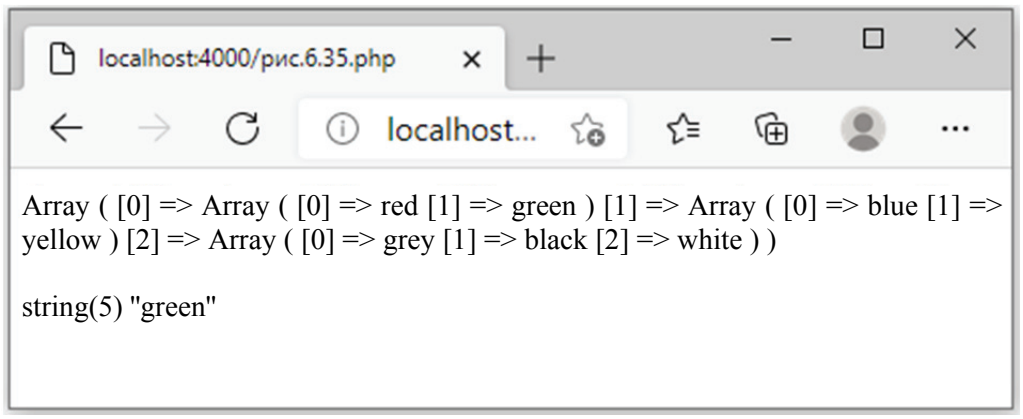

Рис. 6.35. Использование ввода массива одной записью

Отметим, что для просмотра массива в данном примере использовалась функция print_r(). Также для вывода массивов применяется конструкция (цикл) foreach (данный тип цикла для перебора элементов массива будет рассмотрен более подробно в гл. 7).

```
<?php
    $colors = array('красный', 'синий', 'зеленый',
'желтый');

    foreach ($colors as $color) {
       echo "Вам нравится $color?<br>";
    }
?>
```

Результат работы рассмотренного скрипта представлен на рис. 6.36.

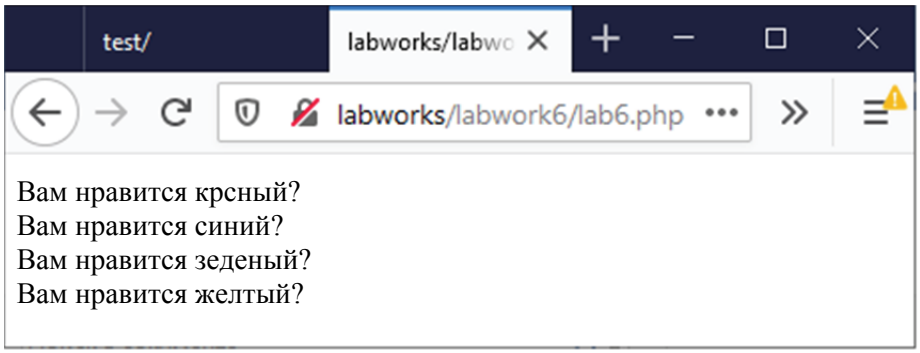

Рис. 6.36. Результат выполнения вывода массива с помощью foreach

Пример 2

Рассмотрим пример сортировки многомерного массива (рис. 6.37) по нескольким столбцам.

volume	edition
67	2
86	1
85	6
98	2
86	6
67	7

Рис. 6.37. Пример массива для сортировки

В примере мы будем сортировать по *volume* в убывающем порядке, а по *edition* – в возрастающем. Для этого воспользуемся функцией array_multisort().

Для начала введем массив.

```
$data[] = array('volume' => 67, 'edition' => 2);
$data[] = array('volume' => 86, 'edition' => 1);
$data[] = array('volume' => 85, 'edition' => 6);
$data[] = array('volume' => 98, 'edition' => 2);
$data[] = array('volume' => 86, 'edition' => 6);
$data[] = array('volume' => 67, 'edition' => 7);
```

Далее выделим каждый столбец массива в отдельный одномерный массив.

```
$volume  = array_column($data, 'volume');
$edition = array_column($data, 'edition');
```

И наконец сортируем данные по volume по убыванию и по edition по возрастанию (добавляем $data в качестве последнего параметра для сортировки по общему ключу).

```
array_multisort($volume, SORT_DESC, $edition, SORT_ASC, $data);
```

В конце необходимо добавить вывод отсортированного массива на экран. Полный листинг скрипта будет следующим:

```
<?php
     $data[] = array('volume' => 67, 'edition' => 2);
     $data[] = array('volume' => 86, 'edition' => 1);
     $data[] = array('volume' => 85, 'edition' => 6);
     $data[] = array('volume' => 98, 'edition' => 2);
     $data[] = array('volume' => 86, 'edition' => 6);
     $data[] = array('volume' => 67, 'edition' => 7);
     $volume  = array_column($data, 'volume');
     $edition = array_column($data, 'edition');
     array_multisort($volume,    SORT_DESC,    $edition,
SORT_ASC, $data);

     foreach ($data as $value) {
          echo '<p>';
          print_r($value);
          echo '</p>';
     }
?>
```

Результат работы рассмотренного скрипта представлен на рис. 6.38.

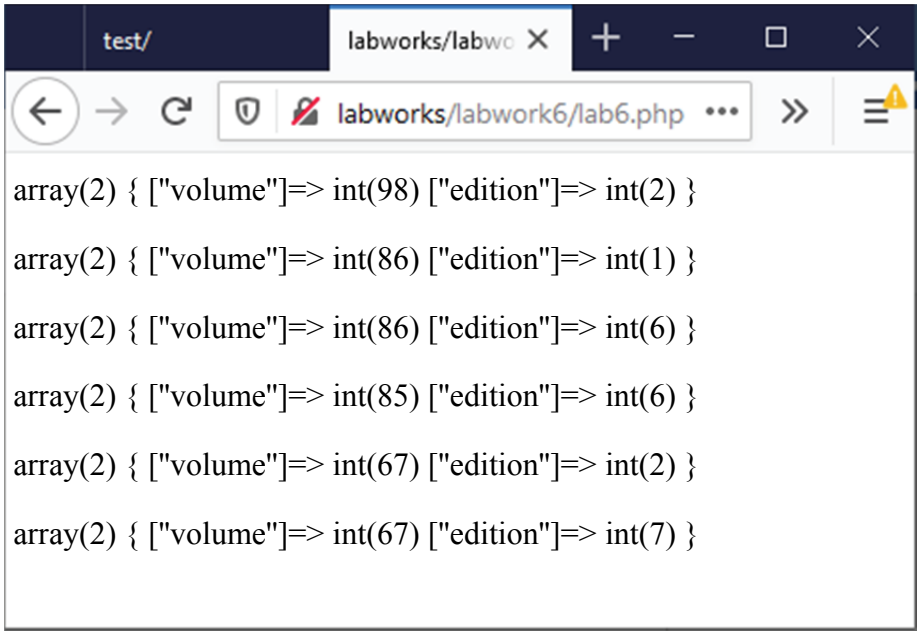

Рис. 6.38. Результат выполнения скрипта для сортировки

Пример 3

Пусть имеется набор предложений, разделенных точками. Разбить текст на массив предложений и найти предложение с

наибольшим количеством слов. Для перебора предложений массива можно использовать циклы.

На начальном этапе посчитаем количество предложений и разобьем текст на массив предложений.

```
$text='Hello. What a beautiful day today. Let\'s go now
for a walk.';
$number_sentences=substr_count($text, '.');
$sentences=explode('.',$text);
```

Отметим, что количество предложений можно было определять как через функцию *substr_count()*, так и путем подсчета количества элементов массива, например функцией *count()*.

Далее посчитаем количество слов в каждом предложении.

```
for ($i=0; $i < $number_sentences; $i++) {
     $number_word[]=str_word_count($sentences[$i]);
}
```

Теперь остается найти номер предложения с максимальным значением количества слов.

```
$max_kol_word=max($number_word);
$key_max_word=array_search($max_kol_word,$number_word)+1;
```

Полный листинг скрипта, решающего поставленную задачу, может быть следующим:

```
<?php
     $text='Hello. What a beautiful day today. Let\'s
go now for a walk.';
     $number_sentences=substr_count($text, '.');
     $sentences=explode('.',$text);
     for ($i=0; $i < $number_sentences; $i++) {
          $num-
ber_word[]=str_word_count($sentences[$i]);
     }
     $max_kol_word=max($number_word);
     $key_max_word=array_search($max_kol_word,$number_word)+1;

     print_r($sentences);
     echo "<br>Наибольше количество слов <b>$max_kol_word</b>
в <b>".$key_max_word.'</b>-м предложении.';
?>
```

Результат выполнения скрипта представлен на рис. 6.39.

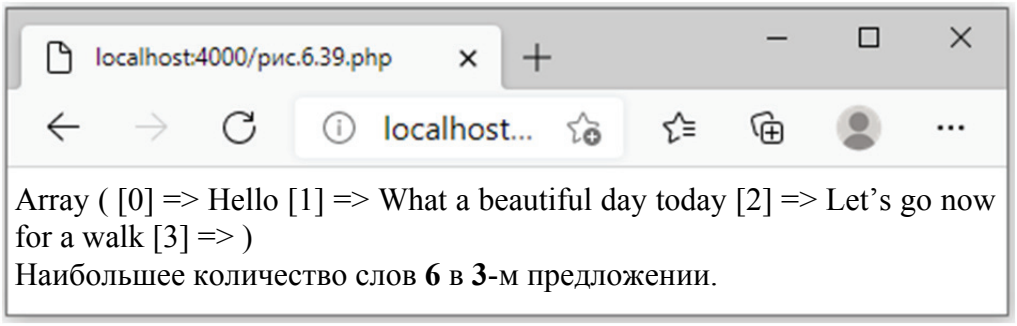

Рис. 6.39. Результат выполнения скрипта для поиска наибольшего (по количеству слов) предложения

Отметим, что решение данной задачи является типичным примером сочетания как функций для работы со строками, так и функций для работы с массивами.

Лабораторная работа № 6

Задания на лабораторную работу будут носить похожий на рассмотренные примеры характер и выдаваться индивидульно преподавателем. Напомним, что выполнение и демострация всех лабораторных работ предполагается в пределах структуры, представленной в лабораторной работе № 1.

ГЛАВА 7

УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ ЯЗЫКА PHP

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

В PHP существуют шесть основных групп управляющих конструкций:

1) условные операторы;
2) циклы;
3) конструкции выбора;
4) конструкции возврата значений;
5) конструкции включений;
6) функции.

7.1. Условные операторы

Условные операторы являются, пожалуй, наиболее распространенной группой конструкций во всех алгоритмических языках программирования. Рассмотрим основные условные операторы языка PHP.

7.1.1 Конструкция if. Синтаксис конструкции if аналогичен конструкции if в языке C:

```
<?PHP
    if (логическое выражение) оператор;
?>
```

Согласно выражениям PHP, конструкция if содержит логическое выражение. Если логическое выражение истинно (true), то оператор, следующий за конструкцией if, будет исполнен, а если логическое выражение ложно (false), то следующий за if оператор исполнен не будет. Приведем примеры.

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

В следующем примере если переменная $*a* не равна нулю, будет выведена строка «значение *a* истинно (true)»:

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

В представленном примере, если переменная $*a* равна нулю, будет выведена строка «значение *a* ложно (false)».

```
<?PHP
    $a=False;
    if (!$a) echo "значение a ложно (false) ";
?>
```

Результаты выполнения скрипта представлены на рис. 7.1.

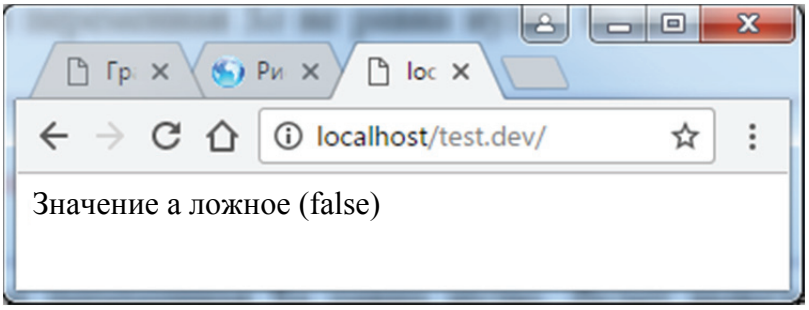

Рис. 7.1. Простейший пример использования if

Часто на практике будет необходим блок операторов, который будет выполняться при определенном условном критерии. В таком случае эти операторы необходимо поместить в фигурные скобки {...}. Приведем пример.

```
<?PHP
   $a=5;
   $b=1;
   if ($a > $b) {
      echo "a больше b";
      $c = $a-$b;
      echo "<br>", "разность равна $c";
   }
?>
```

Результаты выполнения скрипта представлены на рис. 7.2.

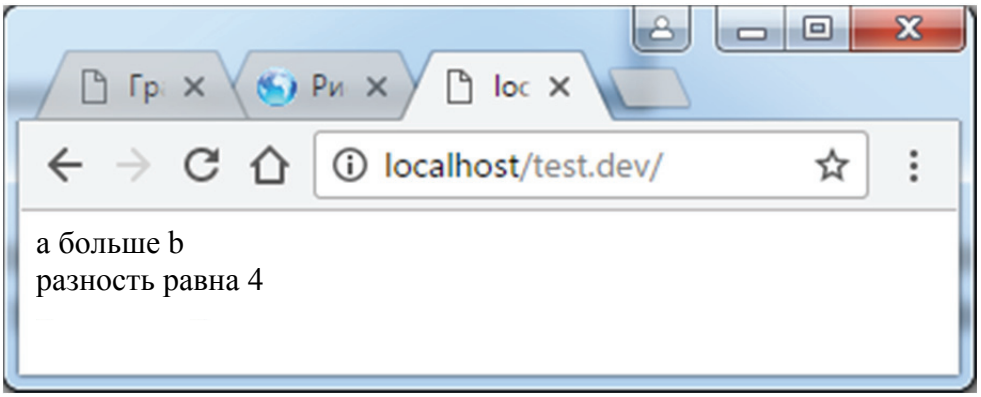

Рис. 7.2. Результаты использования if с блоком операторов

Приведенный пример выведет сообщение «*a* больше *b*», если $*a* > $*b*, а затем будет высчитана и выведена разница между ними. Заметим, что данные операторы выполняются в теле конструкции if.

7.1.2. Конструкция else. Часто возникает потребность исполнения операторов не только в теле конструкции if, если выполнено какое-либо условие конструкции if, но и в случае, если условие конструкции if не выполнено. В данной ситуации нельзя обойтись без конструкции else. В целом такая конструкция будет называться конструкцией if-else. Синтаксис конструкции if-else следующий:

```
if (логическое_выражение)
   инструкция_1;
else
   инструкция_2;
```

Действие конструкции if-else следующее: если логическое выражение истинно, то выполняется инструкция 1, а иначе – инструкция 2. Как и в любом другом языке, конструкция else может опускаться – при получении должного значения просто ничего не делается.

Если инструкция 1 или инструкция 2 должны состоять из нескольких команд, то они как всегда заключаются в фигурные скобки.

```
<?PHP
    $a=5;
    $b=6;
    if ($a > $b) {
       echo "a больше, чем b";
    } else {
       echo "a НЕ больше, чем b";
    }
?>
```

Результаты выполнения скрипта представлены на рис. 7.3.

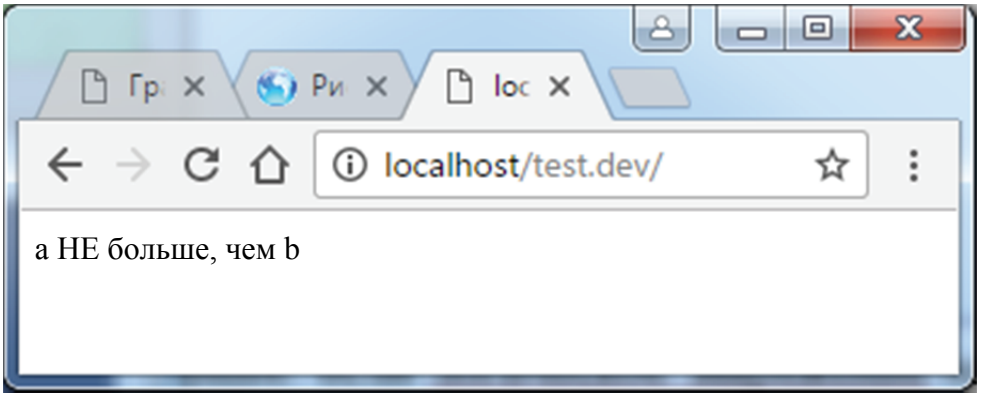

Рис. 7.3. Результаты использования if-else

Конструкция if-else имеет еще один альтернативный синтаксис:

```
if (логическое_выражение):
    команды;
elseif(другое_логическое_выражение):
    другие_команды;
else:
    иначе_команды;
endif
```

Приведем пример.

```
<?PHP
    $a=5;
    $b=6;
    if ($a > $b) :
        echo "a больше, чем b";
    else :
        echo "a НЕ больше, чем b";
    endif
?>
```

Обратите внимание на расположение двоеточия (:)! Если его пропустить, будет сгенерировано сообщение об ошибке. И еще, как обычно блоки elseif и else можно опускать.

7.1.3. Конструкция elseif – это комбинация конструкций if и else, расширяющая условную конструкцию if-else. Приведем синтаксис конструкции elseif.

```
if (логическое_выражение_1)
    оператор_1;
elseif (логическое_выражение_2)
    оператор_2;
else
    оператор_3;
```

Практический пример использования конструкции elseif.

```
<?PHP
   $a=5;
   $b=5;
   if ($a > $b) {
     echo "a больше, чем b";
   } elseif ($a == $b) {
     echo "a равен b";
   } else {
     echo "a меньше, чем b";
   }
?>
```

Результаты выполнения скрипта представлены на рис. 7.4.

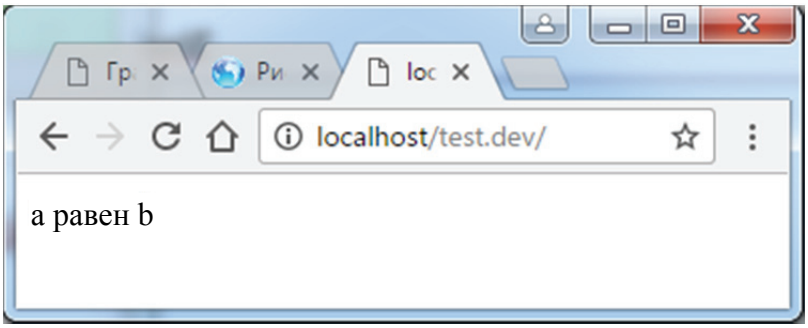

Рис. 7.4. Результаты использования elseif

Конструкция elseif не очень удобна для дальнейшего чтения кода, поэтому применяется не так часто.

7.1.4. Тернарный оператор. Помимо этого, в PHP есть еще один оператор, который представляет собой сокращенную форму конструкции *if-else* – это тернарный оператор. Он возвращает разные результаты в зависимости от того, выполнено условие или нет. В общем виде его использование выглядит следующим образом:

```
условие ? результат_если_true : результат_если_false
```

Рассмотрим применение на простейшем примере нахождения модуля без использование встроенной функции.

```
<?PHP
     $x = -2;
     $mod = $x >= 0 ? $x : -$x;
     echo "Модуль числа $x равен: " . $mod;
?>
```

Результаты выполнения скрипта представлены на рис. 7.5.

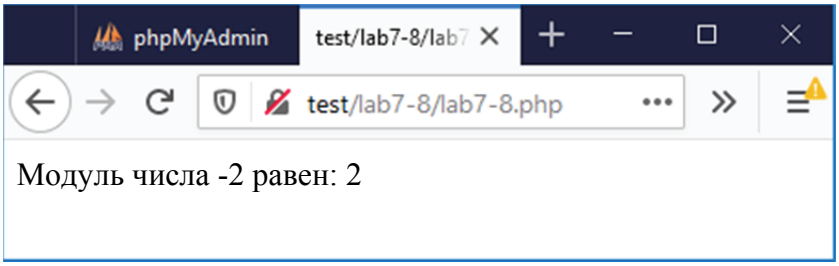

Рис. 7.5. Результаты использования тернарного оператора

Отметим, что тернарный оператор позволил записать условие $\$x >= 0$ и варианты выбора прямо в строку присваивания.

7.2. Циклы

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

Циклы дают возможность повторять определенное (и даже неопределенное – когда работа цикла зависит от условия) количество раз различные операторы. Данные операторы называются телом цикла, а проход цикла – итерацией.

PHP поддерживает следующие виды циклов:

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

При использовании циклов есть возможность применения операторов break и continue. Первый из них прерывает работу всего цикла, а второй – только текущей итерации.

7.2.1. Цикл с предусловием while. Цикл с предусловием while работает по следующим принципам: сначала вычисляется значение логического выражения. Затем, если значение истинно, выполняется тело цикла, в противном случае – переходим на следующий за циклом оператор.

Синтаксис цикла с предусловием следующий:

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

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

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

Результаты выполнения скрипта представлены на рис. 7.5.

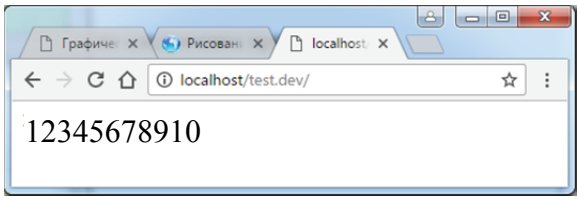

Рис. 7.5. Результаты использования цикла while

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

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

Если увеличить счетчик после выполнения оператора echo, то получим строку 0123456789. В любом случае имеем 10 итераций. Итерация – это выполение операторов внутри тела цикла.

Подобно конструкции условного оператора if можно группировать операторы внутри тела цикла while с помощью следующего альтернативного синтаксиса.

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

Пример использования альтернативного синтаксиса.

```
<?PHP
   $x = 1;
   while ($x <= 10):
```

```
    echo $x;
    $x++;
  endwhile;
?>
```

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

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

После очередной итерации проверяется, истинно ли логическое_выражение, и если это так, управление передается вновь на начало цикла, в противном случае – цикл обрывается.

Альтернативного синтаксиса для do-while разработчики PHP не предусмотрели (видимо, из-за того, что, в отличие от прикладного программирования, этот цикл довольно редко используется при программировании web-приложений). Приведем пример скрипта, показывающего работу цикла с постусловием do-while.

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

Результаты выполнения скрипта представлены на рис. 7.6.

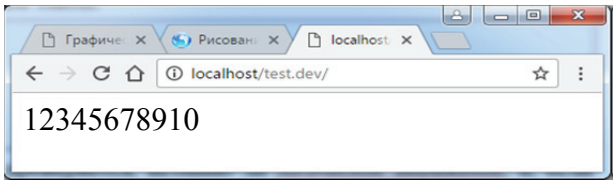

Рис. 7.6. Результаты использования цикла do while

Использование данного вида цикла удобно, в том случае когда мы не знаем, сколько итераций должно быть выполнено, но знаем, что одна итерация выполняется всегда.

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

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

Цикл for начинает свою работу с выполнения инициализирующих команд. Данные команды выполняются только один раз. После этого проверяется условие цикла: если оно истинно (true), то выполняется тело цикла. После того, как будет выполнен последний оператор тела, приступим к командам после итерации. Затем снова проверяется условие цикла. Если оно истинно (true), выполняется тело цикла и команды после итерации и т. д.

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

Результаты выполнения скрипта представлены на рис. 7.7.

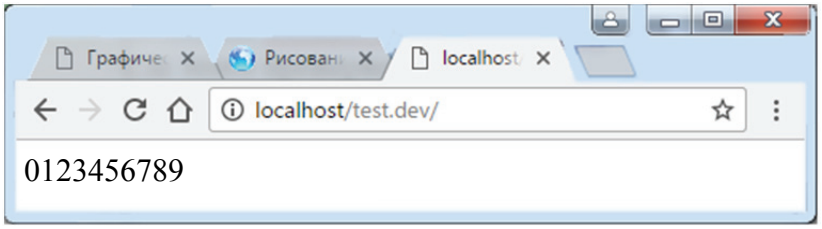

Рис. 7.7. Результаты использования цикла for

Можно второй и третий аргумент цикла совместить, например, следующим образом:

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

Результаты выполнения скрипта представлены на рис. 7.8.

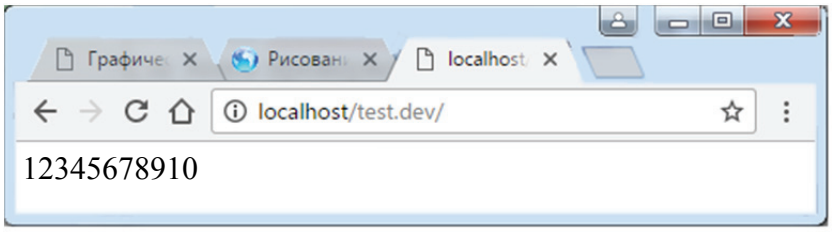

Рис. 7.8. Результаты использования цикла for
с совмещенной записью

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

Если необходимо указать несколько команд, их можно разделить запятыми.

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

Приведем более практичный пример использования нескольких команд в цикле for.

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

Результаты выполнения скрипта представлены на рис. 7.9.

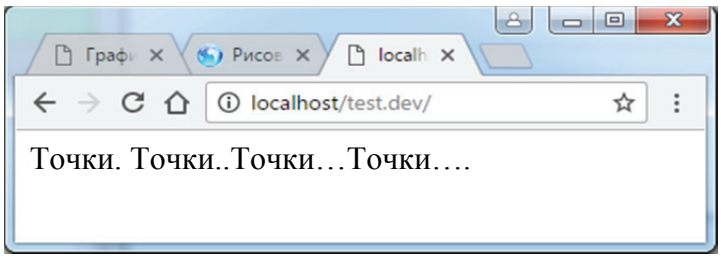

Рис. 7.9. Результаты использования цикла for
с несколькими командами в условии

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

Для цикла for имеется и альтернативный синтаксис.

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

7.2.4. Цикл перебора массивов foreach. Данный цикл предназначен специально для перебора массивов. Синтаксис цикла foreach выглядит следующим образом:

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

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

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

Результаты выполнения скрипта представлены на рис. 7.10.

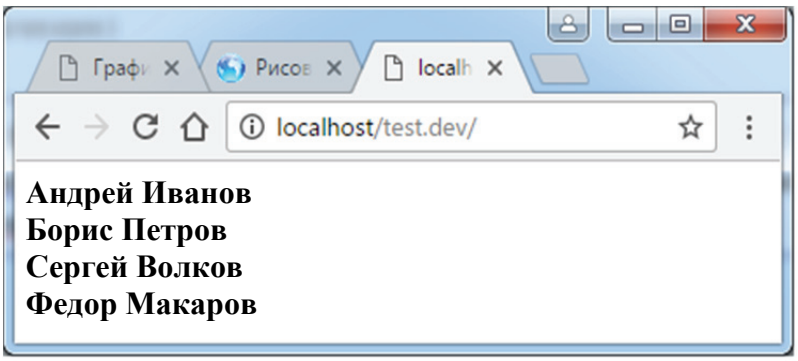

Рис. 7.10. Результаты использования цикла foreach

У цикла foreach имеется и другая форма записи. Она применяется, если нас не интересует значение ключа очередного элемента. Выглядит следующим образом:

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

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

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

Результаты выполнения скрипта представлены на рис. 7.11.

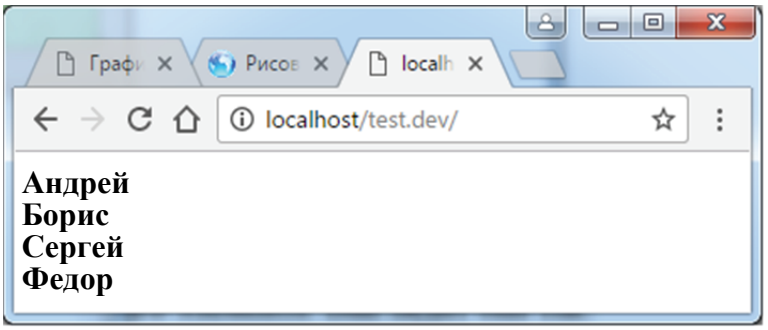

Рис. 7.11. Результаты использования цикла foreach для вывода значений массива

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

7.3. Конструкция break

Очень часто для упрощения логики какого-нибудь сложного цикла удобно иметь возможность его прервать в ходе очередной итерации (например, при выполнении любого условия). Для этого и существует конструкция break, которая осуществляет немедленный выход из цикла. Она может задаваться с одним необязательным параметром – числом, указывающим, из какого вложенного цикла должен быть произведен выход. По умолчанию используется 1, т. е. выход из текущего цикла, но иногда применяются и другие значения. Синтаксис конструкции break следующий:

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

Приведем пример.

```
<?PHP
   $x=0;
   while ($x++<10) {
      if ($x==3) break;
      echo "<b>Итерация $x</b><br>";
   }// Когда $x равен 3, цикл прерывается
?>
```

Результаты выполнения скрипта представлены на рис. 7.12.

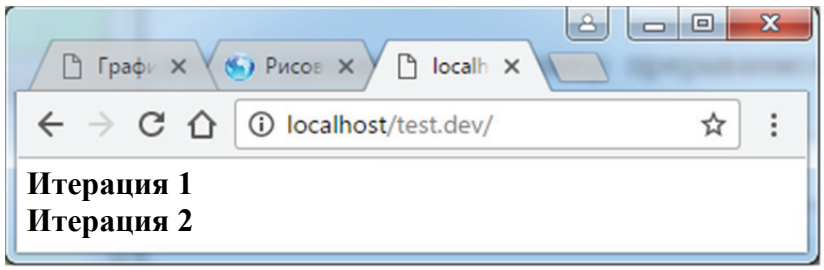

Рис. 7.12. Результаты использования break для прерывания цикла

Если нужно прервать работу определенного (вложенного) цикла, то следует передать конструкции break параметр – номер цикла, например, break(1). Нумерация циклов выглядит следующим образом:

```
for (...) // Третий цикл
{
  for (...) // Второй цикл
  {
     for (...) // Первый цикл
     {
     }
  }
}
```

Рассмотрим пример с двумя циклами.

```
<?PHP
for ($i=1;$i<=4;$i++){
    for ($j=-2;$j<=15;$j++){
        echo "i={$i}","j=$j","<br>";
        if ($j>=$i) break(2);
    }
echo "<br>";
}
?>
```

Результаты выполнения скрипта представлены на рис. 7.13.

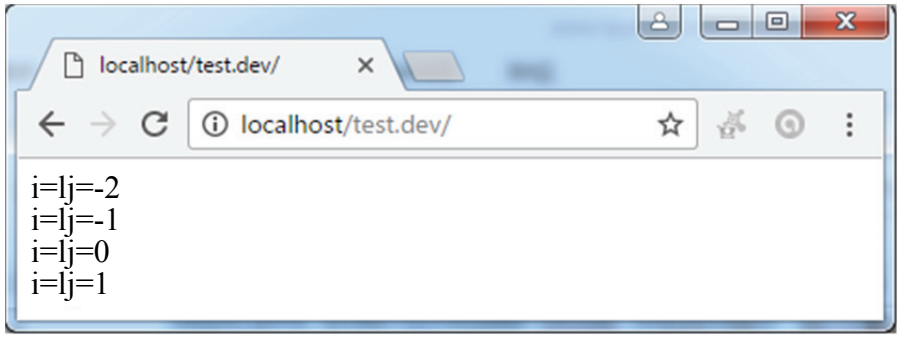

Рис. 7.13. Результаты использования break
для прерывания внешнего цикла

Обратимся еще к одному примеру прерывания цикла – в данном случае внутреннему с индексом 1 при условии совпадения счетчиков обоих циклов.

```
<?PHP
for ($i=1;$i<=4;$i++){
    for ($j=-2;$j<=15;$j++){
```

```
        echo "i={$i}","j=$j","<br>";
        if ($j>=$i) break(1);
    }
echo "<br>";
}
?>
```

Результаты выполнения скрипта представлены на рис. 7.14.

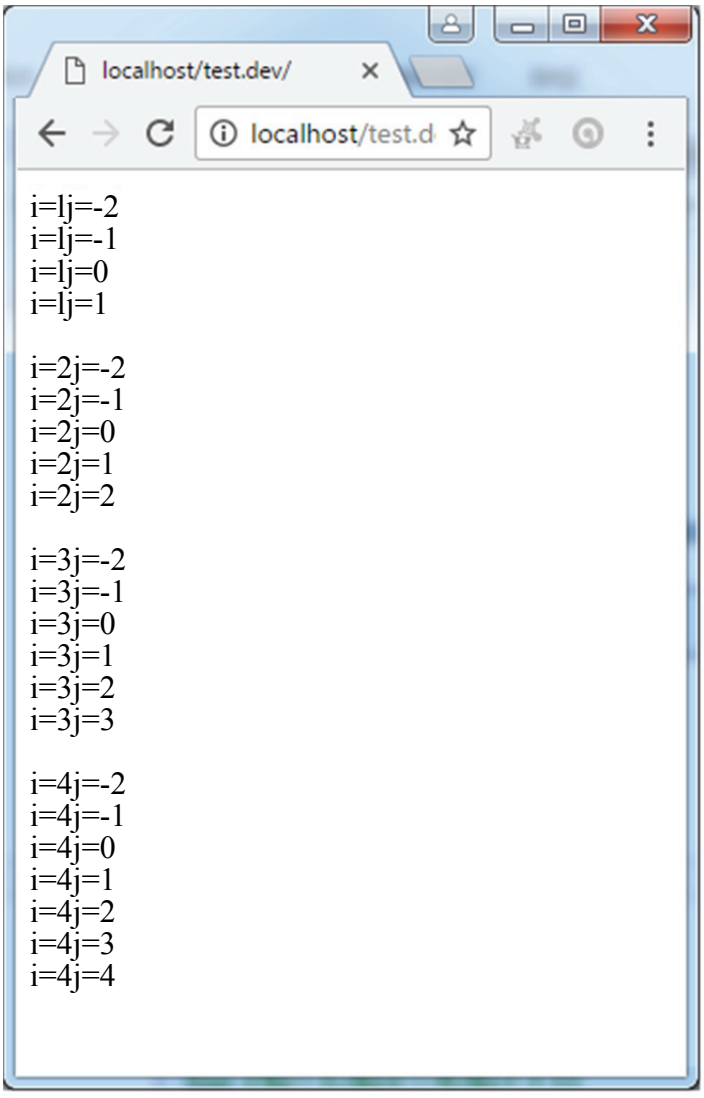

Рис. 7.14. Результаты использования break
для прерывания внутреннего цикла

На рис. 7.14 мы видим, как изменяются значения переменных *i* (внешнего) и *j* (внутреннего) циклов.

7.4. Конструкция continue

Конструкция continue так же, как и break, работает только «в паре» с циклическими конструкциями. Она немедленно завершает текущую итерацию цикла и переходит к новой (конечно, если выполняется условие цикла для цикла с предусловием). Точно так же, как и для break, для continue можно указать уровень вложенности цикла, который будет продолжен по возврату управления.

В основном continue позволяет сэкономить количество фигурных скобок в коде и увеличить его удобочитаемость. Это чаще всего требуется в циклах-фильтрах, когда необходимо перебрать некоторое количество объектов и выбрать из них только те, которые удовлетворяют определенным условиям. Приведем пример использования конструкции continue.

```
<?PHP
$x=0;
while ($x++<5) {
if ($x==3) continue;
echo "<b>Итерация $x</b><br>";
}
// Цикл прервется только на третьей итерации
?>
```

Результаты выполнения скрипта представлены на рис. 7.15.

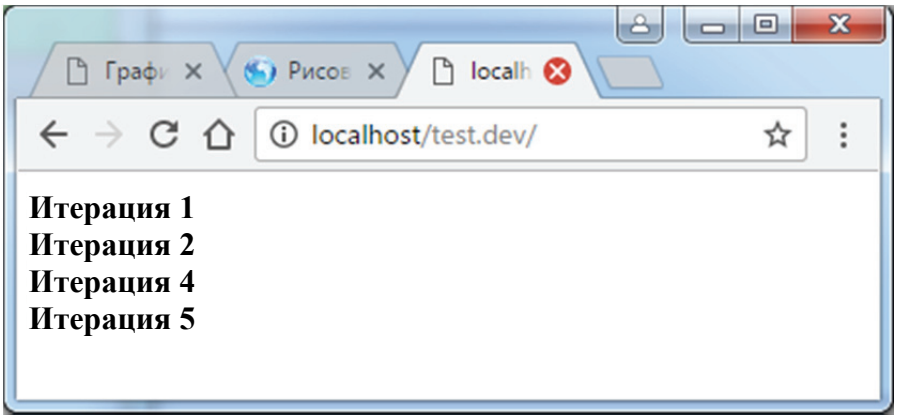

Рис. 7.15. Результаты использования continue

Грамотное использование break и continue дает возможность заметно улучшить «читабельность» кода и количество блоков else.

7.5. Конструкции выбора

Часто вместо нескольких расположенных подряд инструкций if-else целесообразно воспользоваться специальной конструкцией выбора switch-case. Данная конструкция предназначена для выбора действий в зависимости от значения указанного выражения. Конструкция switch-case чем-то напоминает if-else, которая, по сути, является ее аналогом. Конструкцию выбора можно использовать, если предполагаемых вариантов много, например более 5, и для каждого варианта необходимо выполнить специфические действия. В таком случае использование конструкции if-else становится действительно неудобным. Синтаксис конструкции switch-case следующий:

```
switch(выражение) {
   case значение1: команды1; [break;]
   case значение2: команды2; [break;]
   . . .
   case значениеN: командыN; [break;]
   [default: команды_по_умолчанию; [break]]
}
```

Опишем принцип работы конструкции switch-case:

1) вычисляется значение выражения;

2) просматривается набор значений. Пусть значение 1 равно значению выражения, вычисленного на первом шаге. Если не указана конструкция (оператор) break, то будут выполнены команды *i*, *i*+1, *i*+2, ... , *N*. В противном случае (есть break) будет выполнена только команда с номером *i*;

3) если ни одно значение из набора не совпало со значением выражения, то выполняется блок default (если он указан).

Приведем примеры использования конструкции switch-case.

```
<?PHP
   $x=1;
   echo "Работает if else", "<br>";
   if ($x == 0) {
      echo "x=0<br>";
   } elseif ($x == 1) {
      echo "x=1<br>";
   } elseif ($x == 2) {
      echo "x=2<br>";
```

```
    }
    // Используем switch-case
    echo "Работает switch", "<br>";
    switch ($x) {
       case 0:
         echo "x=0<br>";
         break;
       case 1:
         echo "x=1<br>";
         break;
       case 2:
         echo "x=2<br>";
         break;
    }
?>
```

Рассмотренный сценарий выводит $x = 1$ дважды (рис. 7.16).

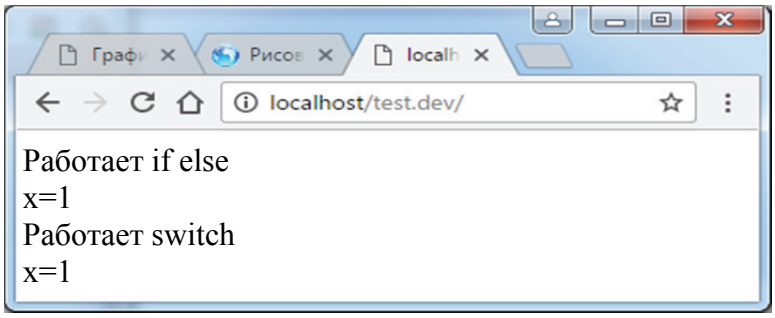

Рис. 7.16. Результаты использования if-else и swich для решения одной и той же задачи

Рассмотрим еще один пример использования конструкции switch-case.

```
<?PHP
    $x="Яблоко";
    switch ($x) {
      case "Яблоко":
         echo "Это Яблоко";
      break;
      case "Груша":
         echo "Это Груша";
      break;
      case "Арбуз":
         echo "Это Арбуз";
      break;
    }
?>
```

Данный скрипт выводит «Это Яблоко» (рис. 7.17).

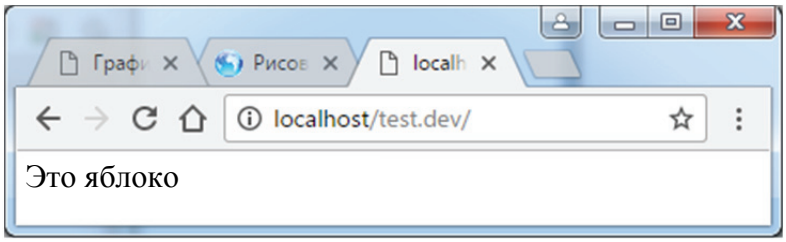

Рис. 7.17. Результаты использования swich

Конструкция switch выполняется поэтапно. Сначала никакой код не исполнен. Только в том случае, когда конструкция case найдена со значением, соответствующим значению выражения switch, PHP начинает исполнять конструкции. PHP продолжает исполнять конструкции до конца блока switch либо пока не встречается оператор break. Без использования конструкций(операторы) break скрипт будет выглядеть следующим образом:

```
<?PHP
  $x=1;
  switch ($x) {
    case 0:
        echo "x=$x<br>";
    case 1:
        echo " x=$x<br>";
    case 2:
        echo " x=$x<br>";
  }
?>
```

Результаты выполнения скрипта представлены на рис. 7.18.

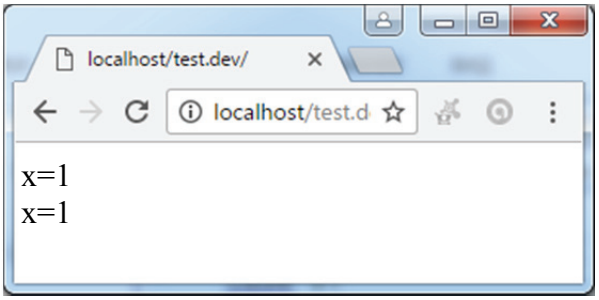

Рис. 7.18. Результаты применения swich
без использования break

Операторный список для case может быть также пуст – в таком случае он просто передает управление в операторный список до следующей конструкции case.

```
<?PHP
   switch ($x) {
   case 0:
   case 1:
   case 2:
     echo "x меньше, чем 3, но не отрицателен";
     break;
   case 3:
     echo "x=3";
   }
?>
```

В случае, когда ни одно значение из набора не совпало со значением выражения, выполняется блок default (если он указан). Например:

```
<?PHP
   $x=3;
   switch ($x) {
     case 0:
        echo "x=0";
     break;
     case 1:
        echo "x=1";
     break;
     case 2:
        echo "x=2";
     break;
     default:
     echo "x не равен 0, 1 или 2";
  }
?>
```

Данный скрипт выводит «*x* не равен 0, 1 или 2», поскольку переменная $\$x = 3$ (рис. 7.19).

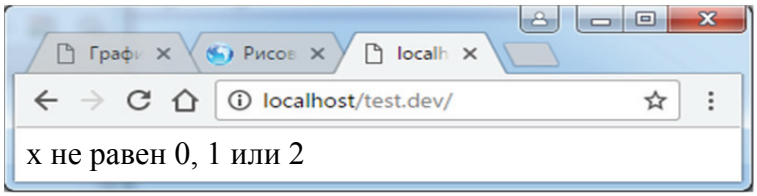

Рис. 7.19. Результаты использования swich
с параметром default

Конструкция switch-case также имеет альтернативный синтаксис.

```
switch(выражение):
   case значение1: команды1; [break;]

   . . .
   case значениеN: командыN; [break;]
   [default: команды_по_умолчанию; [break]]
endswitch;
```

Практический пример использования альтернативного синтаксиса для конструкции switch-case.

```
<?PHP
   $x=3;
   switch ($x):
     case 0:
        echo "x=0";
     break;
     case 1:
        echo "x=1";
     break;
     case 2:
        echo "x=2";
     break;
     default:
        echo "x не равен 0, 1 или 2";
   endswitch;
?>
```

Очевидно, что данный скрипт выводит «*x* не равен 0, 1 или 2», так как $*x*=3.

7.6. Конструкции возврата значений return

Конструкция return возвращает значения преимущественно из пользовательских функций как параметры функционального запроса. При вызове return исполнение пользовательской функции прерывается и возвращаются определенные значения.

Если конструкция return будет вызвана из глобальной области определения (вне пользовательских функций), то скрипт также завершит свою работу и также будут возвращены определенные значения.

Конструкция return используется преимущественно для возврата значений пользовательскими функциями.

Возвращаемые значения могут быть любого типа, в том числе списки и объекты. Возврат приводит к завершению выполнения функции и передаче управления обратно к той строке кода, в которой данная функция была вызвана.

Приведем пример использования конструкции return для возврата значений типа integer.

```
<?PHP
   function retfunct()
   {
     return 7;
   }
   echo retfunct();   // выводит '7'.
?>
```

Пример возврата конструкцией return массивов.

```
<?PHP
function numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = numbers();
echo $zero, "<br>";
echo $one, "<br>";
echo $two;

?>
```

Результаты выполнения скрипта представлены на рис. 7.20.

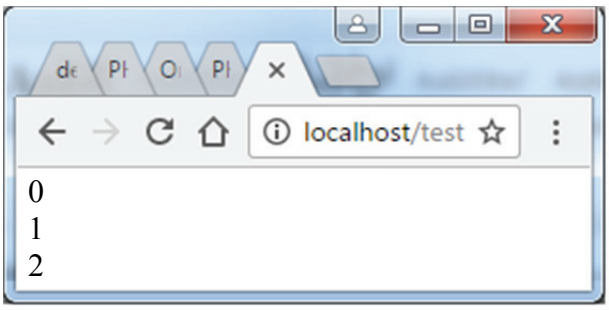

Рис. 7.20. Результаты использования return

В примере приведен вариант вывода значений с помощью списка.

7.7. Конструкции включений

Конструкции включений позволяют собирать PHP-программу (скрипт) из нескольких отдельных файлов.

В PHP существуют две основные конструкции включений: require и include.

7.7.1. Конструкция включений require. Данная конструкция позволяет включать файлы в сценарий PHP до его исполнения сценария PHP. Общий синтаксис require следующий:

```
require имя_файла;
```

При запуске (именно при запуске, а не при исполнении!) программы интерпретатор просто заменит инструкцию на содержимое файла имя файла (этот файл может также содержать сценарий на PHP, обрамленный как обычно тегами <? и?>). Причем сделает он это непосредственно перед запуском программы (в отличие от include, который рассматривается ниже). Это бывает довольно удобно для включения в вывод сценария различных шаблонных страниц html-кодом. Приведем пример объединения двух файлов (header.html и footer.html) в третьем (script.php). Первый файл – header.html.

```
<html>
<head><title>It is a title</title></head>
<body bgcolor=green>
```

Второй файл – footer.html.

```
h3>Hello!!!</h3>
</body></html>
```

Третий – файл script.php

```
<?PHP
require "header.html";
// Сценарий выводит само тело документа
require "footer.html";
?>
```

Таким образом, конструкция require позволяет собирать сценарии PHP из нескольких отдельных файлов, которые могут быть как html-страницами, так и PHP-скриптами. Результат выполнения скрипта script.php представлен на рис. 7.21

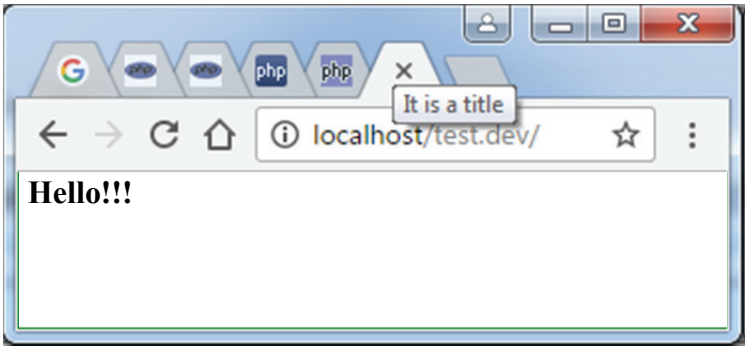

Рис. 7.21. Результаты использования конструкции require

Конструкция require поддерживает включения удаленных файлов.

```
<?PHP

// Следующий пример не работает, поскольку пытается вклю-
чить локальный файл
    require 'file.php?foo=1&bar=2';
    // Следующий пример работает
require 'http://www.example.com/file.php?foo=1&bar=2';
?>
```

Данная конструкция require позволяет включать удаленные файлы, если такая возможность включена в конфигурационном файле PHP.

7.7.2. Конструкция включений include. Такая конструкция также предназначена для включения файлов в код сценария PHP.

В отличие от конструкции require, она позволяет включать файлы в код PHP-скрипта во время выполнения сценария. Синтаксис конструкции include выглядит следующим образом:

```
include имя_файла;
```

Поясним принципиальную разницу между конструкциями require и include на конкретном практическом примере. Создадим

10 файлов с именами 1.txt, 2.txt и т. д. до 5.txt. Содержимое этих файлов – просто десятичные цифры 1, 2, ..., 5 (по одной цифре в каждом файле). Создадим следующий сценарий PHP:

```
<?PHP
   // Создаем цикл, в теле которого конструкция include
   for($i=1; $i<=5; $i++) {
   include "$i.txt";
}
// Включили десять файлов: 1.txt, 2.txt, 3.txt ... 5.tx
t
// Результат - вывод 12345
?>
```

Все файлы располагаются в одной папке (рис. 7.22).

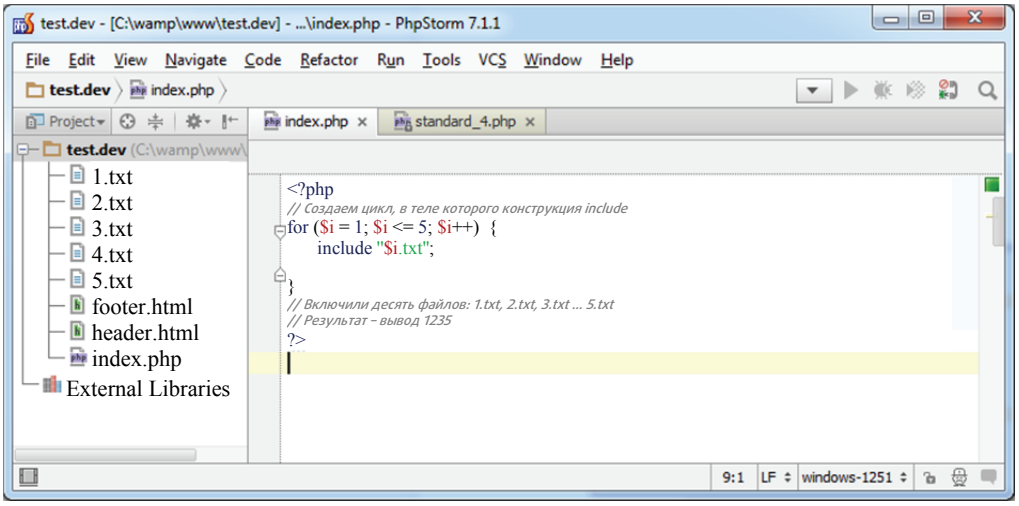

Рис. 7.22. Пример использования конструкции include

В результате получим значения 12345, как показано на рис. 7.23.

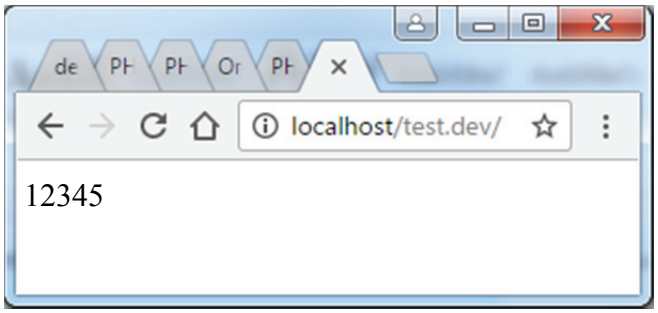

Рис. 7.23. Результат использования конструкции include

Проанализировав результат, можем сделать вывод, что каждый из файлов был включен по одному разу прямо во время выполнения цикла. Если поставить теперь вместо include require, то сценарий сгенерирует критическую ошибку (fatal error).

PHP преобразует сценарий во внутреннее представление, анализируя его строки по очереди, пока не доходит до конструкции include. Дойдя до include, PHP прекращает транслировать сценарий и переключается на указанный в include файл. Таким образом, ввиду подобного поведения транслятора быстродействие сценария снижается, особенно при большом количестве включаемых с помощью include файлов. С require таких проблем не возникает, поскольку файлы с помощью require включаются до выполнения сценария, т. е. на момент трансляции файл уже включен в сценарий.

Таким образом, целесообразнее использовать конструкцию require там, где не требуется динамического включения файлов в сценарий, а конструкцию include – только с целью динамического включения файлов в код PHP-скрипта.

Конструкция include поддерживает включения удаленных файлов (начиная с версии PHP 4.3.0). Например:

```
<?PHP

// Следующий пример не работает, поскольку пытается вклю
чить локальный файл
   include 'file.php?foo=1&bar=2';
   // Следующий пример работает
include 'http://www.example.com/file.php?foo=1&bar=2';
?>
```

7.7.3. Конструкции однократного включения require_once и include_once. В больших PHP-сценариях инструкции include и require применяются очень часто. Поэтому становится довольно сложно контролировать, как бы случайно не включить один и тот же файл несколько раз, что чаще всего приводит к ошибке, которую сложно обнаружить.

В PHP предусмотрено решение данной проблемы. Используя конструкции однократного включения require_once и include_once, можно быть уверенным, что один файл не будет включен дважды.

Работают конструкции однократного включения require_once и include_once так же, как и require и include соответственно. Разница состоит в том, что перед включением файла интерпретатор проверяет, включен ли указанный файл ранее или нет. Если да, то файл не будет включен вновь.

Конструкции однократных включений, такие как require_once и include_ince, также позволяют включать удаленные файлы (если такая возможность включена в конфигурационном файле PHP).

7.8. Ссылки

В PHP нет такого понятия, как указатель, но все же существует возможность создавать ссылки на другие переменные. Есть две разновидности ссылок: жесткие и символические, часто называемые просто ссылками. Ссылки в PHP − это средство доступа к содержимому одной переменной под разными именами. Они не похожи на указатели языка C и не являются псевдонимами таблицы символов. В PHP имя переменной и ее содержимое − это разные вещи, поэтому одно содержимое может иметь разные имена.

7.8.1. Жесткие ссылки в PHP. Жесткая ссылка представляет собой просто переменную, которая является синонимом другой переменной. Многоуровневые ссылки, т. е. ссылка на ссылку на переменную, как это можно делать, например, в Perl, не поддерживаются, поэтому не стоит воспринимать жесткие ссылки серьезнее, чем синонимы.

Для создания жесткой ссылки нужно использовать оператор & (амперсанд).

```
<?PHP
   $a=10;
   $b = &$a; // теперь $b - то же самое, что и $a
   echo "b=$b, a=$a"; // Выводит: "b=10, a=10"
   echo "<br>";
   $b=0; // на самом деле $a=0
   echo "b=$b, a=$a"; // Выводит: "b=0, a=0"
?>
```

Результат выполнения скрипта представлен на рис. 7.24.

Ссылаться можно не только на переменные, но и на элементы массива. Этим жесткие ссылки выгодно отличаются от символических. Приведем пример.

```
<?PHP
   $A=array('a' => 'aaa', 'b' => 'bbb');
   $b=&$A['b']; // теперь $b - то же, что и элемент с
индексом 'b' массива
   echo $A['b'],"<br>";
   $b=0;
   echo $A['b'];
?>
```

Результат выполнения скрипта представлен на рис. 7.25.

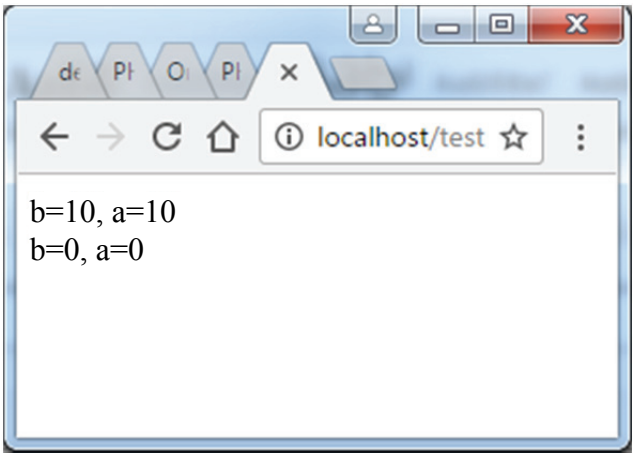

Рис. 7.24. Результат использования жестких ссылок

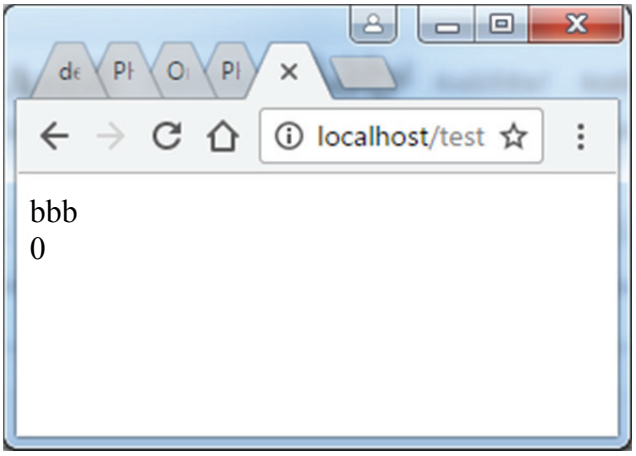

Рис. 7.25. Результат использования жестких ссылок на элемент массива

Впрочем, элемент массива, для которого планируется создать жесткую ссылку, может и не существовать.

```
<?PHP
    $A=array('a' => 'aaa', 'b' => 'bbb');
    $b=&$A['c']; // теперь $b - то же, что и элемент с
индексом 'c' массива
    echo "Элемент с индексом 'c': (".$A['c'].")";
?>
```

Результат выполнения скрипта представлен на рис. 7.26.

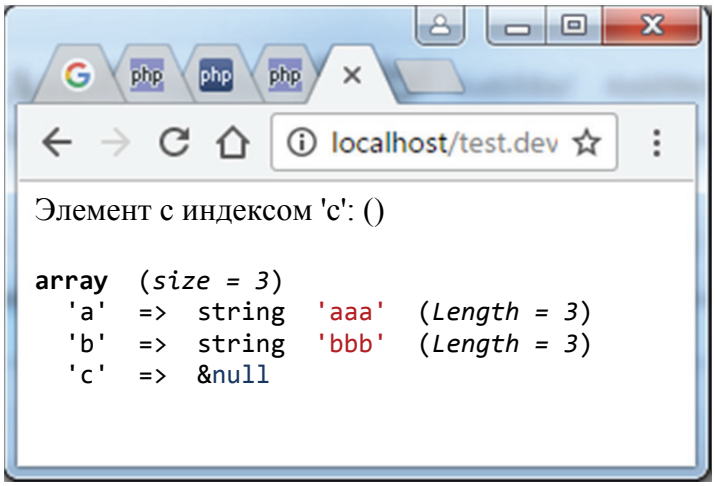

Рис. 7.25. Результат использования жестких ссылок на несуществующий элемент массива

В результате выполнения рассмотренного скрипта, хотя ссылке $*b* и не было ничего присвоено, в массиве $*A* создастся новый элемент с ключом *c* и пустым значением. Жесткая ссылка на самом деле не может ссылаться на несуществующий объект, а если делается такая попытка, то объект создается.

Данный вывод можно проверить, убрав или закомментировать строку, в которой создается жесткая ссылка. В итоге будет выведено сообщение о том, что элемент с ключом *c* не определен в массиве $*A*.

Жесткие ссылки удобно применять при передаче параметров пользовательской функции и возврате значения из нее.

7.8.2. Символические ссылки. Символическая ссылка – это всего лишь строковая переменная, хранящая имя другой переменной.

Для доступа к значению переменной, на которую ссылается символическая ссылка, необходимо применить дополнительный знак $ перед именем ссылки. Рассмотрим следующий простой пример:

```
<?PHP
    $a=10;
    $b=20;
    $c=30;
    $p="a"; // или $p="b" или $p="c" (присваиваем $p
имя другой переменной)
    echo $$p,"<br>"; // выводит переменную, на которую
ссылается $p, т. е. $a
    echo $p,"<br>";

    $$p=100; // присваивает $a значение 100
    echo $a;
?>
```

Результат выполнения скрипта представлен на рис. 7.27.

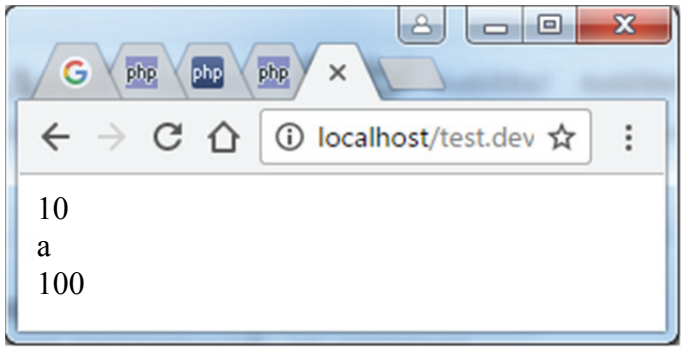

Рис. 7.27. Результат использования символических ссылок

Видно, что для того чтобы использовать обычную строковую переменную как ссылку, нужно перед ней поставить еще один символ $. Это говорит интерпретатору о необходимости взять не значение самой переменной, например $*p*, а значение переменной, имя которой хранится в переменной $*p*.

7.8.3. Передача значений по ссылке с использованием функций. Можно передавать переменные в пользовательскую функцию по ссылке, если необходимо разрешить функции моди-

фицировать аргументы, используемые в основной части кода при вызове функции. Именно в таком случае пользовательская функция сможет изменять аргументы. Приведем простейший пример. Отметим, что применение функций будет подробно рассмотрено в подразд. 7.9.

```
<?PHP
    function f(&$var)
    {
       $var++;
    }

    $a=5;
    f($a);
    echo $a; // $a здесь равно 6
?>
```

Результат выполнения скрипта представлен на рис. 7.28.

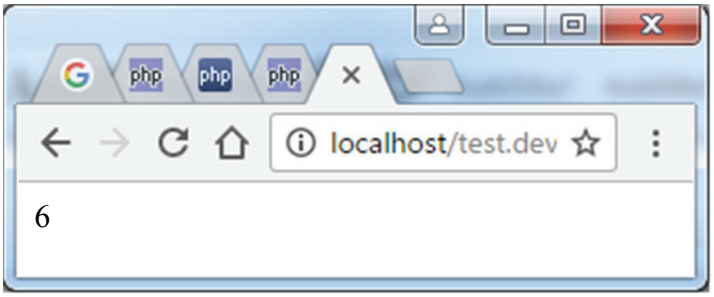

Рис. 7.28. Результат использования жестких ссылок при вызове функции

Необходимо отметить, что в вызове функции отсутствует знак ссылки – он есть только в определении функции. Этого достаточно для корректной передачи аргументов по ссылке.

7.8.4. Удаление ссылок. При удалении ссылки просто разрывается связь имени и содержимого переменной. Это не означает, что содержимое переменной, на которую ссылались, будет разрушено.

```
<?PHP
   $a = 1;
   $b =&$a;
```

```
    echo "a=$a",", b=$b";
    unset($a);
    echo "<br>","a=$a";
    echo "<br>","b=$b";
    $a=2;
    echo "<br>","a=$a",", b=$b";

?>
```

Результат выполнения скрипта представлен на рис. 7.29.

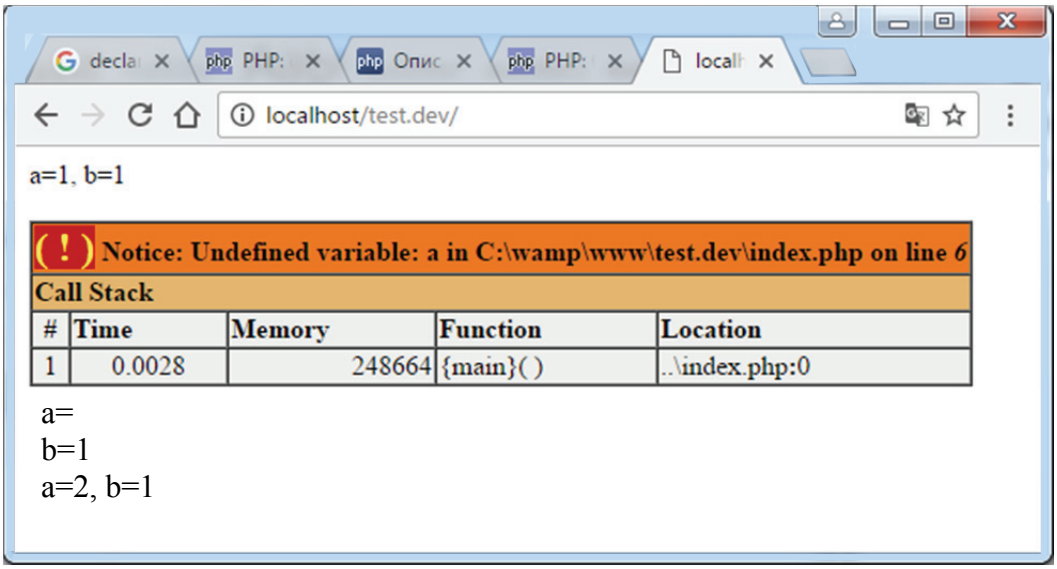

Рис. 7.29. Результат использования жестких ссылок при удалении переменной

Как видно из рис. 7.29, в результате выполнения кода при удалении переменной $*a* возникала ошибка, а значение переменной $*b* сохранилось.

И все же, жесткая ссылка – не абсолютно точный синоним объекта, на который она ссылается. Дело в том, что оператор unset(), выполненный для жесткой ссылки, не удаляет объект, на который она ссылается, а всего лишь разрывает связь между ссылкой и объектом. Жесткая ссылка и переменная (объект), на которую она ссылается, совершенно равноправны, но изменение одной влечет изменение другой.

Оператор unset() разрывает связь между объектом и ссылкой, но объект удаляется только тогда, когда на него никто уже не ссылается.

7.9. Пользовательские функции

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

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

В PHP такими подпрограммами являются пользовательские функции.

Помимо встроенных функций PHP, часто возникает необходимость создания пользовательских функций, выполняющих определенные задачи.

7.9.1. Создание пользовательских функций. Пользовательская функция может быть объявлена в любой части программы (скрипта) до места ее первого использования. И не нужно никакого предварительного объявления, как в других языках программирования, в частности в С. Преимущества применяемого в PHP подхода в следующем.

Дойдя до определения пользовательской функции, транслятор проверит корректность определения и выполнит трансляцию определения функции во внутреннее представление, но транслировать сам код он не будет. И это правильно − зачем транслировать код, который, возможно, вообще не будет использован. Синтаксис объявления функций следующий:

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

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

Требования, предъявляемые к именам функций:

− имена функций могут содержать русские буквы, но давать функциям имена, состоящие из русских букв, не рекомендуется;

− имена функций не должны содержать пробелов;

− имя каждой пользовательской функции должно быть уникальным. При этом необходимо помнить, что регистр при объявлении функций и обращении к ним не учитывается, т. е., например, функции funct() и FUNCT() имеют одинаковые имена;

− функциям можно давать такие же имена, как и переменным, только без знака $ в начале имен.

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

Приведем пример применения пользовательских функций.

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

В рассмотренном примере функция funct возвращает с помощью конструкции return число 777. Возвращенное функцией значение присваивается глобальной переменной $*a*, а затем оператор echo выводит значение переменной $*a* в браузер. В результате в браузере (рис. 7.30) будет отображено число 777.

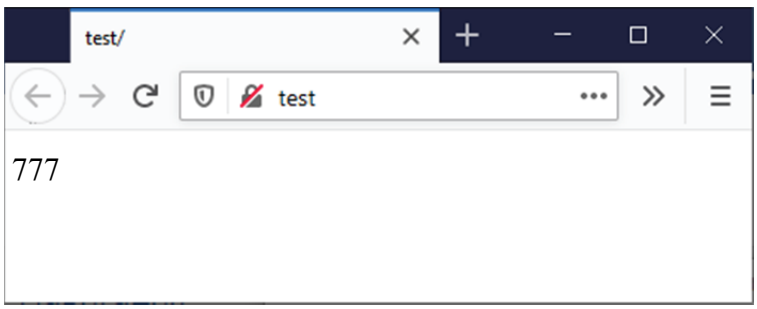

Рис. 7.30. Результат применения пользовательской функци

Пользовательские функции позволяют сократить объем кода и избежать появления ошибок.

7.9.2. Особенности пользовательских функций PHP.

Перечислим особенности пользовательских функций в PHP:

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

В PHP программисту дана достаточно высокая свобода при создании пользовательских функций. Так, например, в отличие от языка C++, в пользовательских функциях доступны параметры по умолчанию.

Недостатки пользовательских функций PHP:

1. Невозможность объявления локальных функций. В PHP нельзя объявить локальную функцию, как это можно сделать в других языках программирования. Попросту говоря, нет возможности создать функцию внутри другой функции таким образом, чтобы первая (вложенная) функция была видна только во второй функции. В PHP вложенная функция будет доступна всей программе (скрипту), а значит не будет локальной. Рассмотрим пример.

```
<?PHP
    function first_function() {
       echo "<h4>Первая пользовательская функция</h4>";
        function second_function() {
       echo "<h5>Вторая пользовательская функция</h5>";
        }
    }
first_function();
second_function();
?>
```

Результат выполнения скрипта представлен на рис. 7.31.

Как видно из рис. 7.31, обе функции доступны программе. Это говорит о том, что вторая функция не является локальной.

2. Второй недостаток пользовательских функций PHP связан с областью видимости функций. Чтобы разобраться в этой категории,

необходимо пояснить, какие переменные являются глобальными, а какие − локальными.

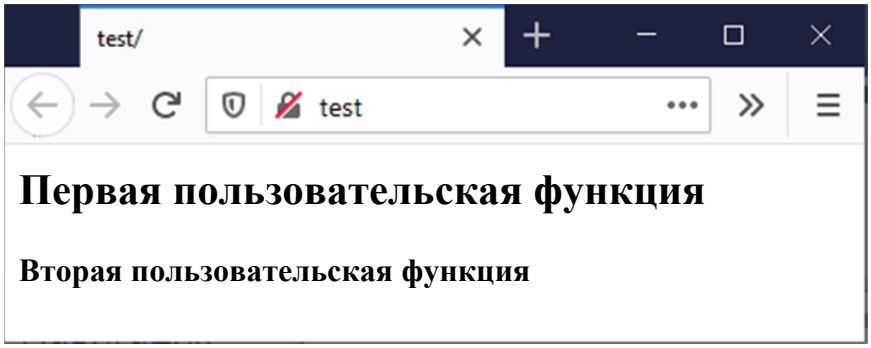

Рис. 7.31. Результат использования вложенных функций

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

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

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

Если же в теле пользовательской функции будет использоваться переменная с именем, идентичным имени глобальной переменной (находящейся вне пользовательской функции), то никакого отношения к глобальной переменной эта локальный переменная иметь не будет. В данной ситуации в пользовательской функции будет создана локальная переменная с именем, идентичным имени глобальной переменной, но доступна данная локальная переменная будет только внутри этой пользовательской функции.

Поясним данный факт на конкретном примере.

```
<?PHP
    $a = 100; /* глобальная область видимости */
    function funct() {
        $a = 70;
/* ссылка на переменную локальной области видимости */
        echo "<h4>$a</h4>";
    }
    funct();
    echo "<h2>$a</h2>";
?>
```

Сценарий выведет сперва 70, а затем 100 (рис. 7.32).

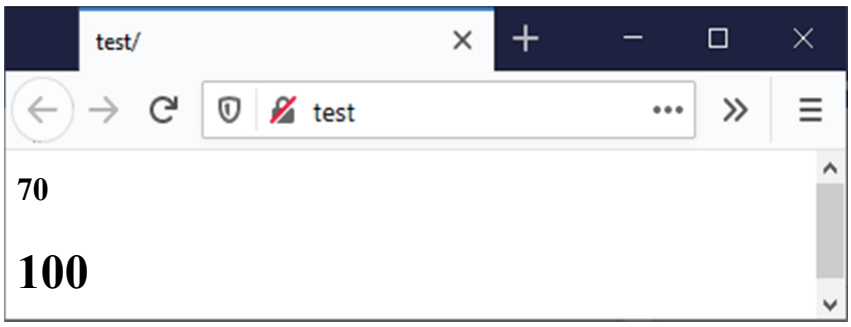

Рис. 7.32. Результат использования
локальных и глобальных переменных

При выходе из пользовательской функции, в которой была объявлена локальная переменная, эта переменная и ее значение уничтожаются.

Основное достоинство локальных переменных – отсутствие непредвиденных побочных эффектов, связанных со случайной или намеренной модификацией глобальной переменной. Рассмотрим еще один пример.

```
<?PHP
    $a = 1; /* глобальная область видимости */
    function Test(){
       echo $a; /* ссылка на переменную локальной обла-
сти видимости */
    }
    Test();
?>
```

Данный скрипт не сгенерирует никакого вывода, поскольку выражение echo указывает на локальную переменную $*a*, а в пределах локальной области видимости ей не было присвоено значение.

Подход к области видимости в PHP отличается от языка C в том, что глобальные переменные в C автоматически доступны функциям, если только они не были перезаписаны локальным определением.

В PHP существует специальная инструкция global, позволяющая пользовательской функции работать с глобальными переменными. Рассмотрим данный принцип на конкретных примерах.

```
<?PHP
    $a = 1;
    $b = 2;
    function Sum(){
         global $a, $b;
         $b = $a + $b;
    }
    Sum();
    echo $b;
?>
```

Вышеприведенный скрипт выведет значение 3 (рис. 7.33). После определения $*a* и $*b* внутри функции как global все ссылки на любую из этих переменных будут указывать на их глобальную версию.

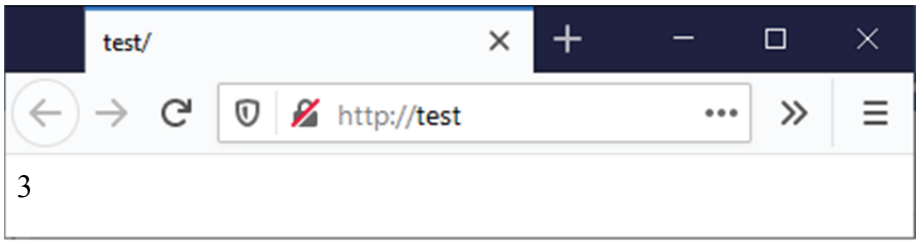

Рис. 7.33. Результат использование
в функциях пользователя инструкции global

Необходимо отметить, что не существует никаких ограничений на количество глобальных переменных, которые могут обрабатываться пользовательскими функциями.

Второй способ доступа к переменным глобальной области видимости – использование специального, определяемого PHP-массива $GLOBALS.

Рассмотрим пример использования $GLOBALS вместо global.

```
<?PHP
    $a = 1;
    $b = 5;
    function Sum(){
         $GLOBALS["c"] = $GLOBALS["a"] + $GLOBALS["b"];
    }
    Sum();
    echo $c;
?>
```

Результат выполнения данного скрипта представлен на рис. 7.34.

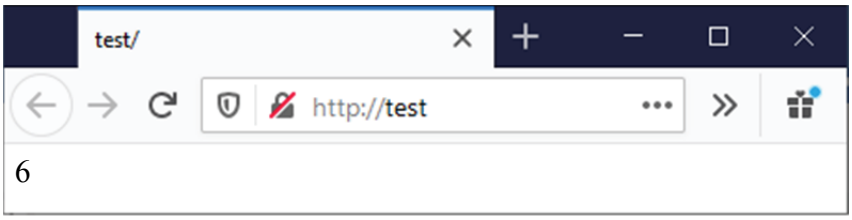

Рис. 7.34. Результаты использования
в функции пользователя массива $GLOBALS

$GLOBALS − это ассоциативный массив, ключом которого является имя, а значением – содержимое глобальной переменной. Обратите внимание, что $GLOBALS существует в любой области видимости.

7.9.3. Передача аргументов пользовательским функциям. При объявлении функции можно указать список параметров, которые могут ей передаваться.

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

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

Передача аргументов по ссылке. Согласно сложившимся традициям, во всех языках программирования есть два вида аргументов функций:

− параметры-значения;

− параметры-переменные.

Функции не могут изменить параметр-значение, т. е. он доступен функции «только для чтения» − она может его использовать, но не более. В качестве параметра-значения необязательно указывать переменную, можно указать само значение, отсюда название – параметр-значение.

По умолчанию аргументы в функцию передаются по значению (это означает, что если вы измените значение аргумента внутри функции, то вне ее значение все равно останется прежним). Приведем пример использования параметра-значения.

```
<?PHP
    function funct($string)
    {
       echo "<h3>Параметр = $string </h3>";
    }

    $str = 777;
    funct(777);
    funct($str);
?>
```

Результат выполнения скрипта представлен на рис. 7.35.

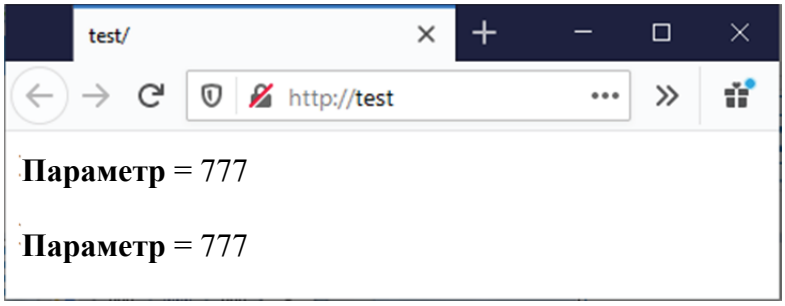

Рис. 7.35. Результат использования параметра-значения в функциях пользователя

В отличие от параметров-значений, параметры-переменные могут быть изменены в процессе работы функции. Тут уже нельзя передавать значение, нужно обязательно передать переменную. В PHP для объявления параметров-переменных используется механизм передачи переменной по ссылке.

Если вы хотите разрешить функции модифицировать свои аргументы, вы должны передавать их по ссылке.

Если вы хотите, чтобы аргумент всегда передавался по ссылке, вы должны указать амперсанд (&) перед именем аргумента в описании функции.

```
<?PHP
    function funct(&$string){
        $string .= ' А эта внутри.';
    }
    $str = 'Эта строка за пределами функции. ';
    echo $str,"<br>";
    funct($str);
    echo $str;
?>
```

Результат выполнения скрипта представлен на рис. 7.36.

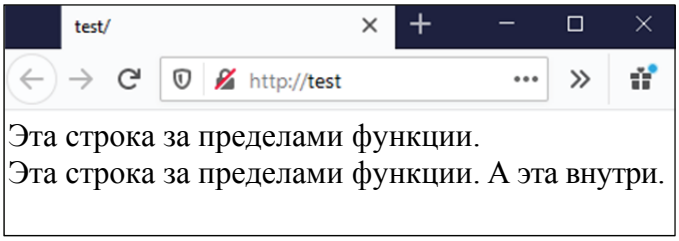

Рис. 7.36. Результат применения параметров-переменных в функциях пользователя

Параметры по умолчанию. При программировании часто возникает необходимость создания функции с переменным числом параметров. Тому есть две причины:

− параметров слишком много, и при этом нет смысла каждый раз указывать их все;

− функции должны возвращать значения разных типов в зависимости от набора параметров.

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

```
<?PHP
    function makecup($type = "Чая")
    {
        return "Сделайте чашечку $type";
    }
    echo makecup();
    echo "<br>";
    echo makecup("Кофе");
?>
```

Результат работы скрипта будет следующим (рис. 7.37):

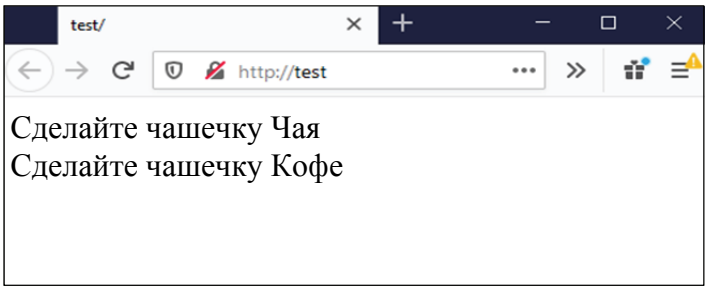

Рис. 7.37. Результаты использования функции пользователя с параметрами по умолчанию

PHP также позволяет использовать массивы и специальный тип NULL в качестве значений по умолчанию. Значение по умолчанию должно быть константным выражением.

Обратите внимание, что все аргументы, для которых установлены значения по умолчанию, должны находится правее аргументов, для которых значения по умолчанию не заданы, в противном случае ваш код может работать не так, как предполагается. Рассмотрим следующий пример:

```
<?PHP

    function makecup($type = "чая", $cond)
    {
        return "Сделайте чашечку $type $cond.";
    }
    echo makecup("горячего");
    // Не будет работать так, как мы могли бы ожидать
?>
```

Результат выполнения скрипта представлены на рис. 7.38.

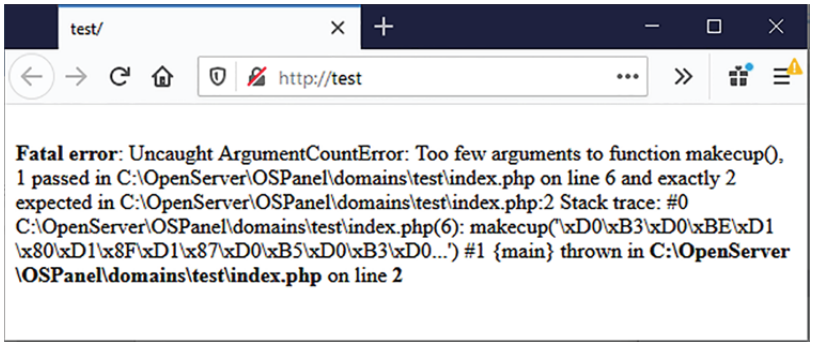

Рис. 7.38. Результаты использования функции пользователя с неправильной последовательностью параметров

Если же выполнить все требования (переменные со значениями по умолчанию должны располагать правее остальных параметров), то получим ожидаемые результаты (рис. 7.39).

```
<?PHP
function makecup($cond, $type = "чая")
    {
        return "Сделайте чашечку $type $cond.";
    }
    echo makecup("горячего");
?>
```

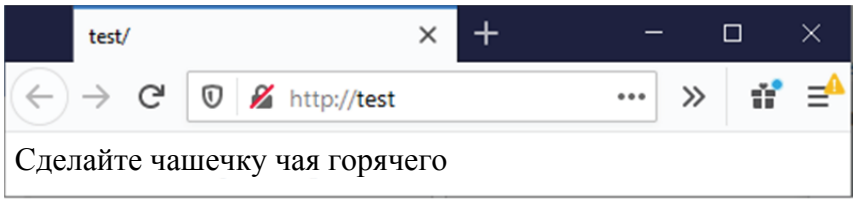

Рис. 7.39. Результаты использования функции пользователя с правильной последовательностью параметров

Отметим, что значения по умолчанию могут быть переданы также и по ссылке.

Переменное число аргументов в функциях. Иногда точно неизвестно, сколько параметров будет передано функции. Специально для такого случая разработчики PHP предусмотрели возможность использования переменного числа аргументов.

Реализация этой возможности достаточно прозрачна и заключается в использовании функций func_num_args(), func_get_arg() и func_get_args().

Рассмотрим возможности рассмотренных стандартных функций:

1. Стандартная функция func_num_args() возвращает количество аргументов, переданных пользовательской функции.

```
<?PHP
    function funct()
    {
         $numargums = func_num_args();
         echo "Количество аргументов : $numargums\n";
    }

    funct(1, 2, 3);
?>
```

Результаты выполнения скрипта представлены на рис. 7.40.

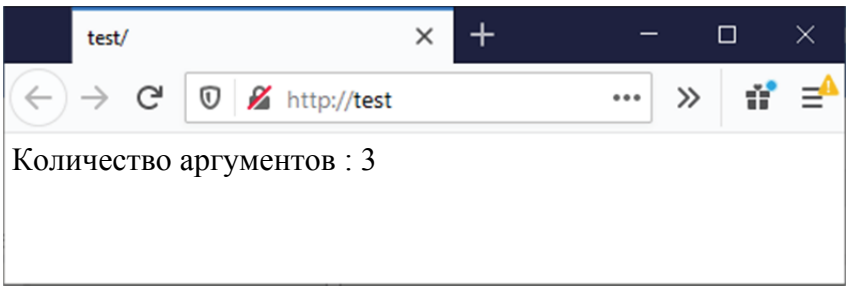

Рис. 7.40. Результат использования функции func_num_args() в функции пользователя

2. Стандартная функция func_get_arg() возвращает элемент из списка переданных пользовательской функции аргументов.

```
<?PHP
    function funct()
    {
         $numargs = func_num_args();
         echo "Колличество аргументов: $numargs<br>";
         if ($numargs >= 2) {
        echo "Второй аргумент : ".func_get_arg(1)."<br>";
          }
    }
    funct(1, 2, 3);
?>
```

Результаты выполнения скрипта представлены на рис. 7.41.

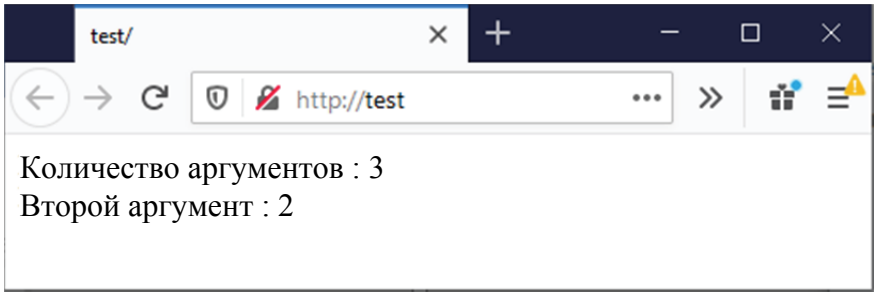

Рис. 7.41. Результат использования функции func_num_args() и func_get_arg() в функции пользователя

3. Стандартная функция func_get_args() возвращает массив аргументов, переданных пользовательской функции.

```
<?PHP
  function funct()
  {
   $numargs = func_num_args();
      echo "Колличество аргументов : $numargs<br>";
      if ($numargs >= 2) {
        echo "Второй аргумент: " . func_get_arg(1) . "<br>";
      }
      $arg_list = func_get_args();
      for ($i = 0; $i < $numargs; $i++) {
        echo "Аргумент $i is: " . $arg_list[$i] . "<br>";
      }
  }
  funct(1, 2, 3);
?>
```

Результаты выполнения скрипта представлены на рис. 7.42.

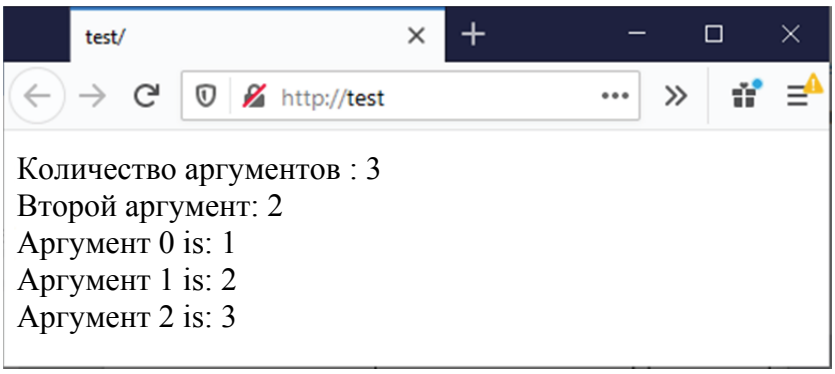

Рис. 7.42. Результат использования функции func_num_args(), func_get_args() и func_get_arg() в функции пользователя

Необходимо отметить, что в данном случае при объявлении пользовательских функций в скобках ничего не пишется (как будто аргументы не передаются).

7.9.4. Статические переменные. Помимо локальных и глобальных переменных, в PHP существует еще один тип переменных − статические.

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

```
<?PHP

   function funct()
   {
      static $a;
      $a++;
      echo "$a";
   }

   for ($i = 0; $i++<10;) funct();
?>
```

Результат выполнения данного сценария с использованием статической переменной представлен на рис. 7.43.

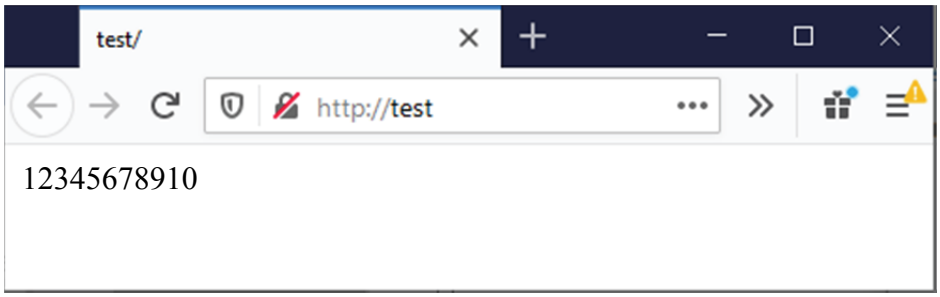

Рис. 7.43. Результат использования статической переменной в функции пользователя

Однако если удалить инструкцию static, будет выведена иная строка (рис. 7.44).

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

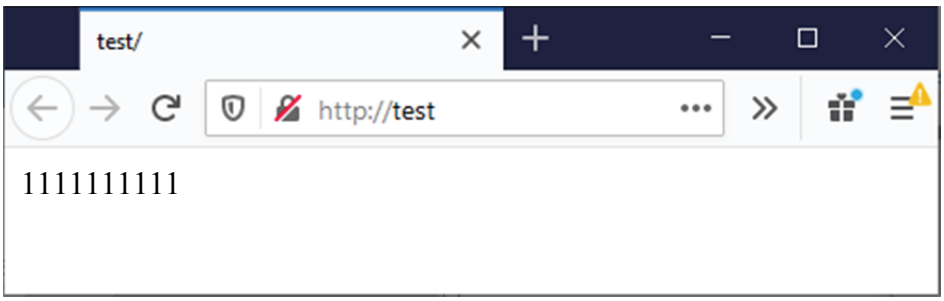

Рис. 7.44. Результат использования переменной без инструкции static в функции пользователя

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

7.9.5. Рекурсивные функции. Рекурсивные функции − это функции, вызывающие самих себя. Такой вызов называется рекурсивным. Рекурсия бывает:

– прямая;
– непрямая.

Рассмотрим пример рекурсивной функции (прямой), используемой при вычислении факториала x!.

```
<?PHP
    function factorial($x)
    {
        if ($x === 0) return 1;
        else return $x*factorial($x-1);
    }
    echo factorial(7);
?>
```

Результат выполнения данного сценария с использованием рекурсии функции представлен на рис. 7.45.

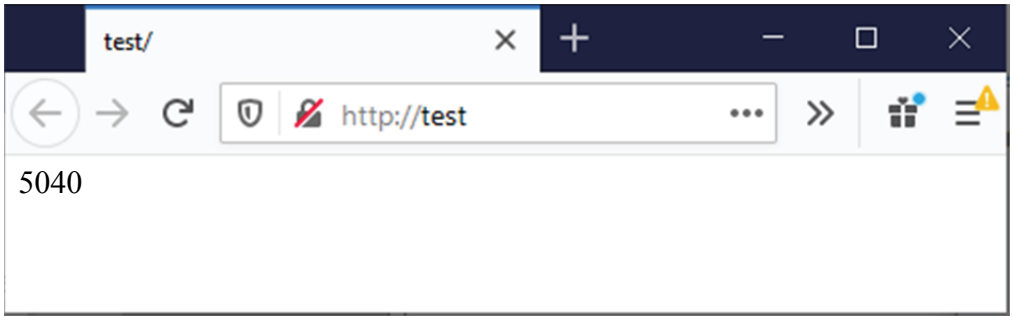

Рис. 7.45. Результат использования прямой рекурсивной функции пользователя

В приведенном примере пользовательская функция factorial() вызывает сама себя, что является прямой рекурсией.

Непрямая рекурсия возникает в случае, когда первая функция вызывает вторую, а вторая – первую.

Отметим, что при создании рекурсивных функций необходимо соблюдать осторожность, стараясь избегать зацикливания.

7.9.6. Условно определяемые функции. PHP позволяет в зависимости от определенных факторов одной и той же функции выполнять различные действия. Рассмотрим следующий пример, который дает возможность выполнять различные действия в зависимости от версии PHP.

```
<?PHP
    $PHPver = PHPversion();
    if ($PHPver[0] === "5")
    {
       function getversion()
       {
            return "Вы используете PHP5";
       }
    }
    if ($PHPver[0] === "4")
    {
        function getversion()
        {
            return "Вы используете PHP4";
        }
    }
    if ($PHPver[0] === "3")
    {
        function getversion()
        {
            return "Вы используете PHP3";
        }
    }
    echo @getversion();
?>
```

Рассмотренный скрипт выводит версию используемого интерпретатора PHP. Одна и та же функция getversion() может возвращать различный результат в зависимости от значения переменной $PHPver.

Практическая часть

Пример 1

Рассмотрим пример реализации простейшего перестановочного шифра: строка, которую необходимо зашифровать, записывается по столбцам в массив с количеством строк, равным количеству букв в ключевом слове (буквы в ключевом слове не могут повторяться). Количество строк будет зависеть от количества символов в шифруемой строке, а ключами строк массива будут выступать

символы ключевого слова. Далее строки в массиве переставляются в соответствии с алфавитным порядком следования ключей. В завершении осуществляется считывание зашифрованной (переставленной) последовательности построчно.

Алгоритм решения этой задачи можно описать следующим набором действий:

1) задаем исходные переменные, хранящие ключ ($key) и текст, который необходимо зашифровать ($word). По ним определяем параметры массива, в который текст и будет в дальнейшем записываться.

```
$key='ivan';
$word='Какой хороший день сегодня!!!';
$columns=intval(strlen($word)/strlen($key))+1;
//определяем количество столбцов массива
$rows=strlen($key);//определяем количество строк массива
```

2) далее записываем по столбцам символы строки в массив. Это можно сделать с использованием цвух циклов for. Операции с отдельными символами строки будем делать с помощью функции substr() или mb_substr(). Последняя является полным аналогом substr(), но для многобайтовых строк, что позволяет зачастую избежать проблем с отображением кириллицы.

```
//формирование массива символов
for($j=0;$j<$columns;$j++){
     for ($i=0; $i<$rows;$i++){
         $mass[$key[$i]][]=mb_substr($word,
0,1);//записываем в массив первую букву строки
         $word=mb_substr($word, 1); //удаляем первую
букву в переменной $word (сдвигаем строку)
     }
}//
```

Данную часть задачи можно решить и иным способом – например, ввести переменную-счетчик ($*k*), которая будет выступать в качества номера символа в строке. Записывать символы в массив тогда можно с помощью простой операции прямого доступа к отдельному символу строки $*word*[$*k*]. С каждым шагом цикла переменную-счетчик необходимо увеличивать на 1.

```
//формирование массива символов
$k=0;
for($j=0;$j<$columns;$j++){
   for ($i=0; $i<$rows;$i++){
       $mass[$key[$i]][]=$word[$k];//записываем в мас-
сив первую букву строки
       $k++;
   }
}//
```

3) сортировку массива по ключам строк выполним с помощью функции ksort();

```
ksort($mass); //сортировка массива
```

4) на последнем этапе выполним обратное преобразование, т. е. считаем с массива с переставленными строками символы и объединим их в одну строковую переменную. Удобнее всего это следать с помощью двух циклов foreach.

```
//формирование зашифрованной строки
foreach ($mass as $value1) {
    foreach ($value1 as $value2) {
        $s_word.=$value2;
    }
}//
```

Решение задачи также необходимо сопроводить выводами соответсвующей информации на различных этапах. Однотипные операции, например вывода массива, можно оформить в виде функции.

Итоговый PHP скрипт может быть следующим:

```
<!DOCTYPE html>
<html>
<head>
    <title>лабораторная работа №7-8 (задача 1)</title>
</head>
<body>
    <h1>
        Лабораторная работа №7-8 (задача 1)
        <hr>
    </h1>
</body>
</html>

<?PHP
```

```
    function out($mass){ //функция, выводящая массив
символов на экран
        foreach ($mass as $key1 => $value1) {
            echo "[$key1] строка массива: ";
            foreach ($value1 as $key2 => $value2) {
                echo $value2.'   ';
            }
            echo "<br>";
        };
    }

    $key='ivan';
    $word='Какой хороший день сегодня!!!';
    echo 'строка, подлежащая перестановке =
'.$word.'<br>';
    echo 'ключевое слово = '.$key.'<br>';

    $columns=intval(strlen($word)/strlen($key))+1;
//определяем количество столбцов массива
    $rows=strlen($key);//определяем количество строк массива

    //формирование массива символов
    for($j=0;$j<$columns;$j++)
    {
        for ($i=0; $i<$rows;$i++)
        {
            $mass[$key[$i]][]=mb_substr($word,
0,1);//записываем в массив первую букву строки
            $word=mb_substr($word, 1); //удаляем первую
букву в переменной $word (сдвигаем строку)
        }
    }//

    echo '<br>Массив с записанной строкой<br><br>';
    out($mass);//вызов фукции вывода массива
    ksort($mass); //сортировка массива
    echo '<br>Отсортированный массив<br><br>';
    out($mass); // вызов фукции вывода отсортированного
по ключам строк массива

    //формирование зашифрованной строки
    foreach ($mass as $value1) {
        foreach ($value1 as $value2)
        {
            $s_word.=$value2;
        }
    }//
    echo '<br>Зашифрованная строка = '.$s_word;
?>
```

Результаты работы скрипта представлены на рис. 7.46.

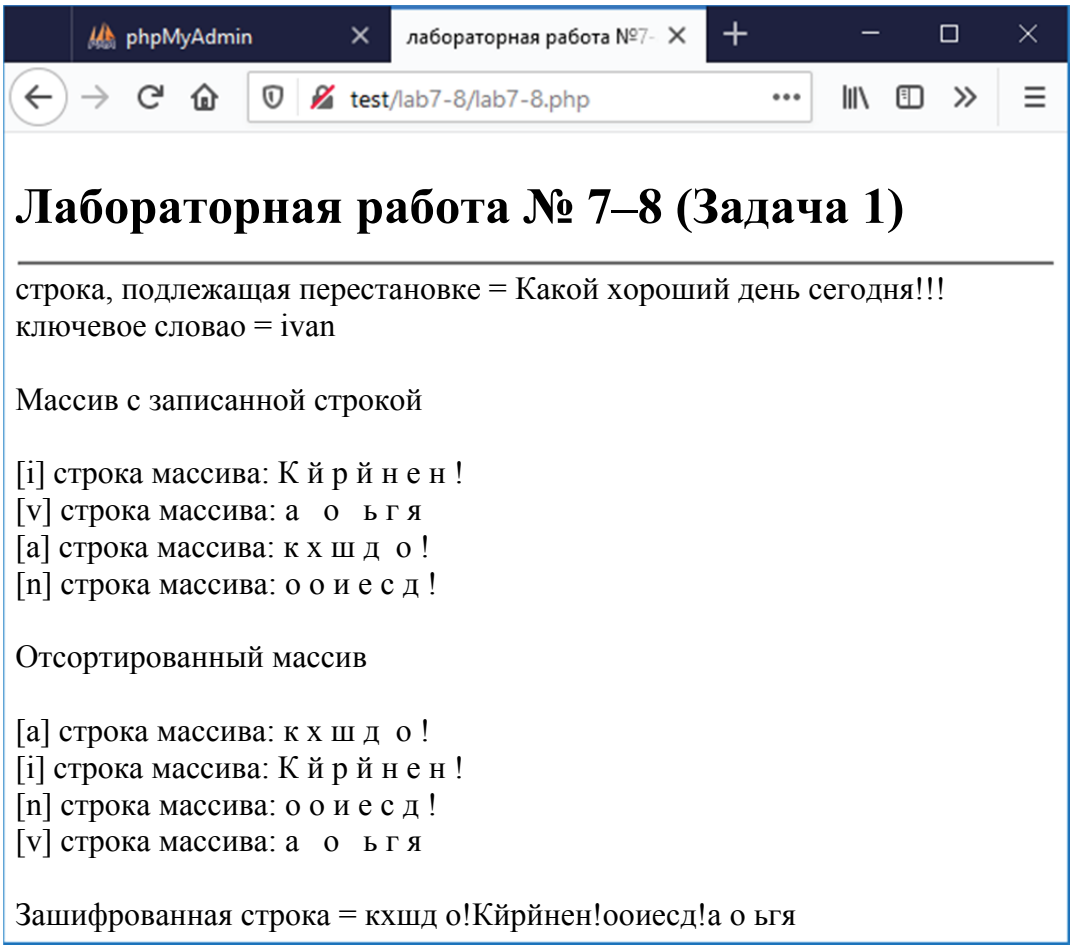

Рис. 7.46. Результат реализации простейшего перестановочного шифра

Пример 2

Рассмотрим пример сортировки двумерного массива, содержащего поле с уникальным идентификатором id, поле с названием категории *name*, а также поле *parent_id*, определяющее уровень вложения («0» в поле *parent_id* означает, что категория является родительской, а все остальные значения, например «1», «2» или «3», указывают подкатегорией какой категии являются.

Общая структура данного массива представлена в таблице. Сортировка массива должна выполняться по двум полям: *id* и *parent_id*. В таком случае при выводе информации сначала будет выводиться первая родительская категория, вслед за ней все ее подкатегории, затем вторая родительская с подкатегориями и т. д. Сам процесс сортировки реализуем в виде отдельной функции.

Отметим, что подобная структура часто встречается при организации меню на web-ресурсах или при организации комментариев и т. д. Для упрощения решения задачи на данном этапе будем предполагать, что возможен только один уровень вложения.

Результаты выведем в виде простого списка категорий с вложенными подкатегориями (таблица).

Пример массива, представляющего меню

id	name	parent_id
1	Web-программирование	0
2	Базы данных	0
3	Обработка изобразительной информации	0
4	Типы данных в PHP	1
5	Конструкции в PHP	1
6	Строки в PHP	1
7	Основы фотографии	3
8	Язык SQL	2
9	Microsoft SQLServer	2
10	Массивы в PHP	1
11	Обработка изображений	3

На начальном этапе необходимо ввести массив.

```
$mass[]=array('id'=>1,'name'=>'Web-
программирование','parent_id'=>0);
$mass[]=array('id'=>2,'name'=>'Базы
данных','parent_id'=>0);
$mass[]=array('id'=>3,'name'=>'Обработка
изобразительной ин-формации','parent_id'=>0);
$mass[]=array('id'=>4,'name'=>'Типы данных в
PHP','parent_id'=>1);
$mass[]=array('id'=>5,'name'=>'Конструкции в
PHP','parent_id'=>1);
$mass[]=array('id'=>6,'name'=>'Строки в
PHP','parent_id'=>1);
$mass[]=array('id'=>7,'name'=>'Основы
фотографии','parent_id'=>3);
$mass[]=array('id'=>8,'name'=>'Язык
SQL','parent_id'=>2);
$mass[]=array('id'=>9,'name'=>'Microsoft SQLServ-
er','parent_id'=>2);
$mass[]=array('id'=>10,'name'=>'Массивы в
PHP','parent_id'=>1);
$mass[]=array('id'=>11,'name'=>'Обработка
изображений','parent_id'=>3);
```

Функция сортировки массива может выглядеть следующим образом:

```
function result_mass_sort($mass)
{
  while (sizeof($mass)>0){
      $mass_result[]=$mass[0];
      unset($mass[0]);
      $mass=array_values($mass);
      $j=sizeof($mass_result);
      for ($i=0; $i<sizeof($mass);++$i)
     {
        if ($mass_result[$j-1]['id']==$mass[$i]['parent_id'])
          {
              $mass_result[]=$mass[$i];
              unset($mass[$i]);
              $mass=array_values($mass);
              $i--;
          }
      }
  }
  return $mass_result;
}
```

Принцип работы данной функции заключается в переборе строк массива *$mass* до тех пор, пока они все не будут перенесы (записаны в новый массив и удалены в старом) в *$mass_result* в соответствии с условием совпадения значений полей *id* и *parent_id* – это реализуется через цикл *while* с условием выхода *(sizeof($mass)>0)*. После каждой операции переноса строки в новый массив индексы в старом массиве ($mass) пересчитываются с помощью функции *array_values();*

Ввиду того, что в данной функции первая строка массива с *parent_id*=0 принимается за начало выводимого списка, то целесообразно предварительно выполнить сортировку с помощью функции array*_multisort()*, например, по столбцам *parent_id* и *name*.

```
$name  = array_column($mass, 'name');
$parent_id = array_column($mass, 'parent_id');
array_multisort($parent_id, $name, SORT_STRING, $mass);
```

Далее остается только вызвать разратобанную функцию сортировки и вывести поле *name* отсортированного массива.

```
$sort_mass=result_mass_sort($mass);

foreach ($sort_mass as $value) {
    if ($value['parent_id']==0){
        echo '<p><b>'.$value['name'].'</b></p>';
    }
    else{
        echo '<p> '.$value['name'].'</p>';
    }
}
```

Полный листинг решения данной задачи будет следующим:

```
<!DOCTYPE html>
<html>
<head>
    <title>лабораторная работа №7-8 (задача 2)</title>
</head>
<body>
    <h1>
        Лабораторная работа №7-8 (задача 2)
        <hr>
    </h1>
</body>
</html>

<?PHP
    function result_mass_sort($mass)
    {
        while (sizeof($mass)>0){
            $mass_result[]=$mass[0];
            unset($mass[0]);
            $mass=array_values($mass);
            $j=sizeof($mass_result);
            for ($i=0; $i<sizeof($mass);++$i)
            {
                if ($mass_result[$j-
1]['id']==$mass[$i]['parent_id'])
                {
                    $mass_result[]=$mass[$i];
                    unset($mass[$i]);
                    $mass=array_values($mass);
                    $i--;
                }
```

```
            }
        }
        return $mass_result;
    }

    $mass[]=array('id'=>1,'name'=>'Web-
программирование','parent_id'=>0);
    $mass[]=array('id'=>2,'name'=>'Базы
данных','parent_id'=>0);
    $mass[]=array('id'=>3,'name'=>'Обработка изобрази-
тельной информации','parent_id'=>0);
    $mass[]=array('id'=>4,'name'=>'Типы данных в
PHP','parent_id'=>1);
    $mass[]=array('id'=>5,'name'=>'Конструкции в
PHP','parent_id'=>1);
    $mass[]=array('id'=>6,'name'=>'Строки в
PHP','parent_id'=>1);
    $mass[]=array('id'=>7,'name'=>'Основы
фотографии','parent_id'=>3);
    $mass[]=array('id'=>8,'name'=>'Язык
SQL','parent_id'=>2);
    $mass[]=array('id'=>9,'name'=>'Microsoft SQLServ-
er','parent_id'=>2);
    $mass[]=array('id'=>10,'name'=>'Массивы в
PHP','parent_id'=>1);
    $mass[]=array('id'=>11,'name'=>'Обработка
изображений','parent_id'=>3);

    $name  = array_column($mass, 'name');
    $parent_id = array_column($mass, 'parent_id');
    array_multisort($parent_id, $name, SORT_STRING,
$mass);

    $sort_mass=result_mass_sort($mass);

    foreach ($sort_mass as $value) {
        if ($value['parent_id']==0){
            echo '<p><b>'.$value['name'].'</b></p>';
        }
        else
        {
            echo '<p> '.$value['name'].'</p>';
        }
    }
?>
```

Результаты выполнения скрипта представлены на рис. 7.47.

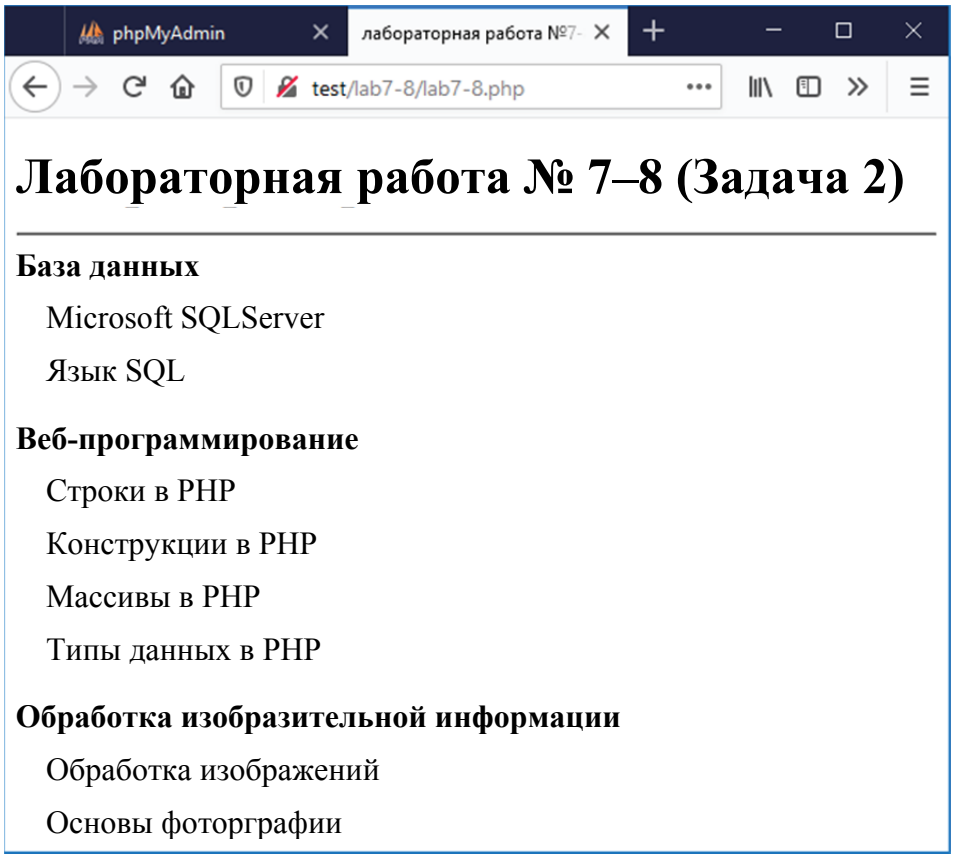

Рис. 7.47. Результат реализации меню

Очевидно, что данная задача может быть решена и другими способами, например более активно задействовав весь набор функции сортировок массивов.

Предложите свой вариант решения.

Пример 3

Рассмотрим решение задачи разработки пользовательской функции по разбиению произвольного текста на предложения, причем пользователь передает в функцию непосредственно текст в виде первого параметра и произвольное число символов-разделителей в виде отдельного параметра каждый их них.

Для решения задачи понадобится разработать две функции. Первая *divide_text()* будет формировать переменную, содержащую переданный текст (*$text*) и массив символов-разделителей *($delimiters)*. Далее она вызывает вторую функцию *multiexplode()*, которая уже непосредственно выполняет деление текста на предложения.

```
function divide_text(){ //функция, делящая текст на
блоки в соответсвии с указанными разделителями
     $numargums = func_num_args();
     $text=func_get_arg(0); //записали в переменную
$text текст, подлежащий разбиению
     $delimiters=func_get_args(); //перенесли в пере-
менную $divide символы-делители текста на блоки
     array_shift($delimiters); //удалили из массива с
символами-делителями сам текст.
     $result=multiexplode( $delimiters, $text);
     return $result;
}
```

Отметим: ввиду того, что количество символов-делителей может быть любым, вторая функция будет рекурсивно вызывать себя же до тех пор, пока не отработает последний символ-делитель, т. е. пока выполняется условие *if($delimiters != NULL)*. Для деления текста на предложения используется функция *explode()*.

```
function multiexplode( $delimiters, $text){
        $arr = explode($delimiters[0],$text);//разбтваем
текст на предложения в соответствии с первым (нулевым по
номеру) символом-делителем
        array_shift($delimiters); //удаляем из массива
символов-делителей отработанный символ-делитель
        if($delimiters != NULL){
            foreach($arr as $key => $sentence){
                 $arr[$key] = multiexplode($delimiters,
$sentence); //рекурсивно вызываем саму же функцию, раз-
бивающую текст на предложения
            }
        }
        return  $arr;
}
```

Далее остается только написать основную часть скрипта, в которой вызывается первая функция *divide_text()* и выводятся на экран результаты.

Полный код данного скрипта будет выглядеть следующим образом:

```
<!DOCTYPE html>
<html>
<head>
    <title>лабораторная работа №7-8 (задача 3)</title>
</head><body>
```

```
    <h1>
        Лабораторная работа №7-8 (задача 3)
        <hr>
    </h1>
</body></html>

<?PHP
    function divide_text(){ //функция, делящая текст на
блоки в соответсвии с указанными разделителями
        $numargums = func_num_args();
        $text=func_get_arg(0); //записали в переменную
$text текст, подлежащий разбиению
        $delimiters=func_get_args(); //перенесли в пе-
ременную $divide символы-делители текста на блоки
        array_shift($delimiters); //удалили из массива
с символами-делителями сам текст.
        $result=multiexplode( $delimiters, $text);
        return $result;
    }
    function multiexplode( $delimiters, $text){
        $arr = explode($delimiters[0],$text);//разбиваем
текст на предложения в соответствии с первым (нулевым
по номеру) символом-делителем
        array_shift($delimiters); //удаляем из массива
символов-делителей отработанный символ-делитель
        if($delimiters != NULL){
            foreach($arr as $key => $sentence){
         $arr[$key] = multiexplode($delimiters, $sentence);
//рекурсивно вызываем саму же функцию, разбивающую
текст на предложения
            }
        }
        return  $arr;
    }
    //основная часть скрипта, из которой вызывается
функция по делению текста на предложения
    $text='PHP - это широко используемый язык сценариев
общего назначения с открытым исходным кодом. PHP - это
язык программирования, специально разработанный для
написания web-приложений (сценариев), исполняющихся на
web-сервере. Синтаксис языка берет начало из C, Java и
Perl. Преимуществом PHP является предоставление web-
разработчикам возможности быстрого создания динамически
генерируемых web-страниц. Также важным преимуществом
языка PHP перед такими языками, как Perl и C, является
возможность создания html-документов с внедренными ко-
мандами PHP. Еще одно преимущество PHP - поддержка ши-
рокого круга баз данных.';
```

```
    $sentences=divide_text($text, '.','!');
    echo '<p><b>'.'Исходный
текст:'.'</b><br>'.$text.'</p><hr>';
    foreach ($sentences as $sentence) {
        echo '<p>'.$sentence[0].'</p>';
    }
?>
```

Результаты выполнения скрипта представлены на рис. 7.48.

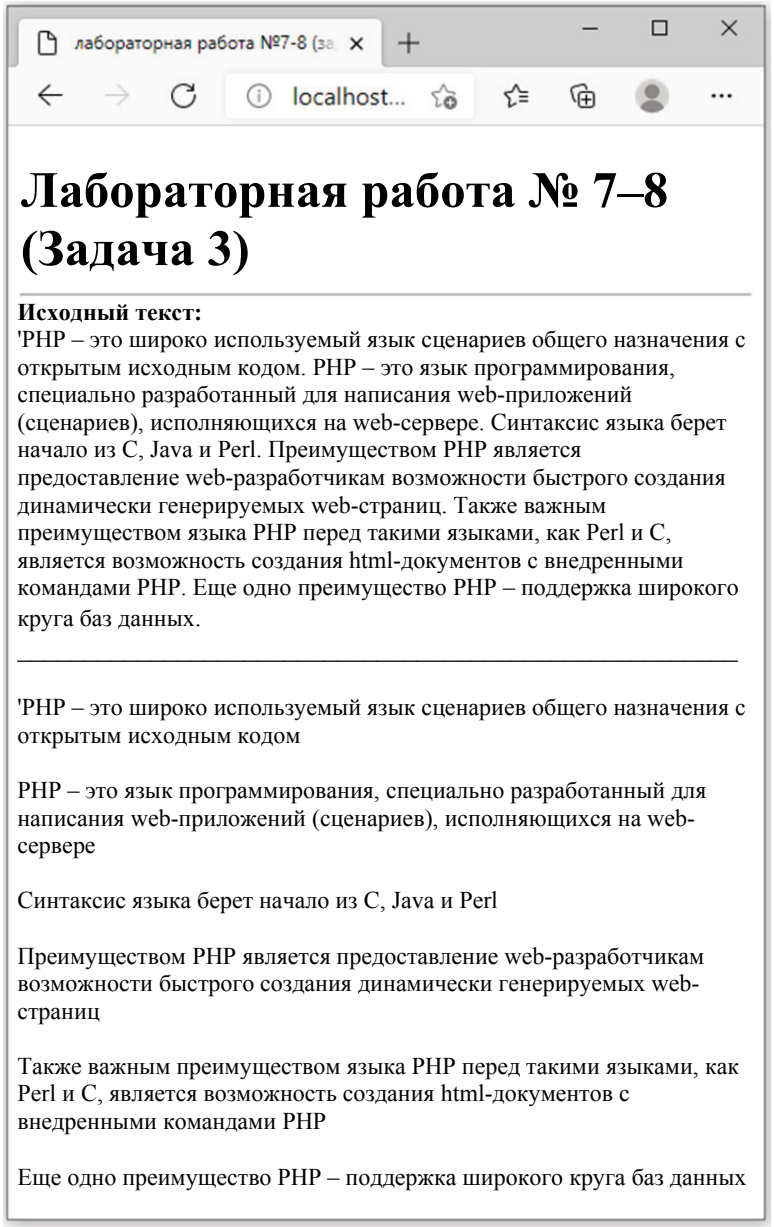

Рис. 7.48. Результат реализации функции
по разбиению текста на предложения

Отметим, что реализацию можно упростить, если пользователь изначально при вызове функции будет передавать символы-разделители не каждый по отдельности, т. е. в виде отдельного параметра, а все вместе в виде массива. В таком случае первая функция, разбивающая принятые параметры на сам текст и массив символов-разделителей, будет не нужна и можно напрямую вызывать вторую функцию.

Лабораторная работа № 7–8

Задания на лабораторную работу будут носить схожий с рассмотренными примерами характер, т. е. будут требовать применения знаний и умений не только по теме «Управляющие конструкции языка PHP», но и по предыдущим главам. Выдаваться задания будут индивидульно преподавателем на занятиях. Напомним, что выполнение и демострация всех лабораторных работ предполагается в пределах структуры, представленной в лабораторной работе № 1.

ПРИЛОЖЕНИЕ А

Результаты использования функций PHP, связанных с типами

Выражение	**gettype()**	**empty()**	**is_null()**	**isset()**	**логическое: *if($x)***
$x = "";	Строка	TRUE	FALSE	TRUE	FALSE
$x = NULL	NULL	TRUE	TRUE	FALSE	FALSE
var $x;	NULL	TRUE	TRUE	FALSE	FALSE
$x неопределена	NULL	TRUE	TRUE	FALSE	FALSE
$x = array();	Массив	TRUE	FALSE	TRUE	FALSE
$x = false;	Логическое	TRUE	FALSE	TRUE	FALSE
$x = true;	Логическое	FALSE	FALSE	TRUE	TRUE
$x = 1;	Целое	FALSE	FALSE	TRUE	TRUE
$x = 42;	Целое	FALSE	FALSE	TRUE	TRUE
$x = 0;	Целое	TRUE	FALSE	TRUE	FALSE
$x = –1;	Целое	FALSE	FALSE	TRUE	TRUE
$x = "1";	Строка	FALSE	FALSE	TRUE	TRUE
$x = "0";	Строка	TRUE	FALSE	TRUE	FALSE
$x = "–1";	Строка	FALSE	FALSE	TRUE	TRUE
$x = "PHP";	Строка	FALSE	FALSE	TRUE	TRUE
$x = "true";	Строка	FALSE	FALSE	TRUE	TRUE
$x = "false";	Строка	FALSE	FALSE	TRUE	TRUE

Сравнение с помощью «==» (сравнение по значению)

Значение	TRUE	FALSE	*1*	*0*	*–1*	*"1"*	*"0"*	*"–1"*	NULL	*array()*	*"PHP"*
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
–1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"–1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
"PHP"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Сравнение с помощью «===» (сравнение по значению и типу)

Значение	TRUE	FALSE	*1*	*0*	*–1*	*"1"*	*"0"*	*"–1"*	NULL	*array()*	*"PHP"*
TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
1	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
–1	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"1"	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"-–"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
NULL	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
array()	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
"PHP"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

ПРИЛОЖЕНИЕ Б

Коды таблицы ASCII для кодировки win-1251

Dec	Hex	Символ
000	00	NOP
001	01	SOH
002	02	STX
003	03	ETX
004	04	EOT
005	05	ENQ
006	06	ACK
007	07	BEL
008	08	BS
009	09	TAB
010	0A	LF
011	0B	VT
012	0C	FF
013	0D	CR
014	0E	SO
015	0F	SI
016	10	DLE
017	11	DC1
018	12	DC2
019	13	DC3
020	14	DC4
021	15	NAK
022	16	SYN
023	17	ETB
024	18	CAN
025	19	EM
026	1A	SUB
027	1B	ESC
028	1C	FS
029	1D	GS
030	1E	RS
031	1F	US
032	20	SP
033	21	!
034	22	"
035	23	#
036	24	$
037	25	%
038	26	&
039	27	'
040	28	(
041	29	)
042	2A	*
043	2B	+
044	2C	,
045	2D	-
046	2E	.
047	2F	/
048	30	0
049	31	1
050	32	2
051	33	3
052	34	4
053	35	5
054	36	6
055	37	7
056	38	8
057	39	9
058	3A	:
059	3B	;
060	3C	<
061	3D	=
062	3E	>
063	3F	?
064	40	@
065	41	A
066	42	B
067	43	C
068	44	D
069	45	E
070	46	F
071	47	G
072	48	H

073	49	I
074	4A	J
075	4B	K
076	4C	L
077	4D	M
078	4E	N
079	4F	O
080	50	P
081	51	Q
082	52	R
083	53	S
084	54	T
085	55	U
086	56	V
087	57	W
088	58	X
089	59	Y
090	5A	Z
091	5B	[
092	5C	\
093	5D	]
094	5E	^
095	5F	_
096	60	`
097	61	a
098	62	b
099	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t

117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL
128	80	Ђ
129	81	Ѓ
130	82	‚
131	83	ѓ
132	84	„
133	85	…
134	86	†
135	87	‡
136	88	€
137	89	‰
138	8A	Љ
139	8B	‹
140	8C	Њ
141	8D	Ќ
142	8E	Ћ
143	8F	Џ
144	90	ђ
145	91	‘
146	92	’
147	93	“
148	94	”
149	95	•
150	96	–
151	97	–
152	98	
153	99	™
154	9A	љ
155	9B	›
156	9C	њ
157	9D	ќ
158	9E	ћ
159	9F	џ
160	A0	–

161	A1	Ў
162	A2	ў
163	A3	Ђ
164	A4	¤
165	A5	Ґ
166	A6	¦
167	A7	§
168	A8	Е
169	A9	©
170	AA	Є
171	AB	«
172	AC	¬
173	AD	–
174	AE	®
175	AF	Ї
176	B0	°
177	B1	±
178	B2	І
179	B3	і
180	B4	ґ
181	B5	µ
182	B6	¶
183	B7	·
184	B8	е
185	B9	№
186	BA	є
187	BB	»
188	BC	ј
189	BD	S
190	BE	s
191	BF	ї
192	C0	А
193	C1	Б
194	C2	В
195	C3	Г
196	C4	Д
197	C5	Е
198	C6	Ж
199	C7	З
200	C8	И
201	C9	Й
202	CA	К
203	CB	Л
204	CC	М

205	CD	Н
206	CE	О
207	CF	П
208	D0	Р
209	D1	С
210	D2	Т
211	D3	У
212	D4	Ф
213	D5	Х
214	D6	Ц
215	D7	Ч
216	D8	Ш
217	D9	Щ
218	DA	Ъ
219	DB	Ы
220	DC	Ь
221	DD	Э
222	DE	Ю
223	DF	Я
224	E0	а
225	E1	б
226	E2	в
227	E3	г
228	E4	д
229	E5	е
230	E6	ж
231	E7	з
232	E8	и
233	E9	й
234	EA	к
235	EB	л
236	EC	м
237	ED	н
238	EE	о
239	EF	п
240	F0	р
241	F1	с
242	F2	т
243	F3	у
244	F4	ф
245	F5	х
246	F6	ц
247	F7	ч
248	F8	ш

249	F9	щ
250	FA	ъ
251	FB	ы
252	FC	ь

253	FD	э
254	FE	ю
255	FF	я

Специальные управляющие символы

Код	Описание
NUL, 00	Null, пустой
SOH, 01	Start Of Heading, начало заголовка
STX, 02	Start of TeXt, начало текста
ETX, 03	End of TeXt, конец текста
EOT, 04	End of Transmission, конец передачи
ENQ, 05	Enquire, прошу подтверждения
ACK, 06	Acknowledgement, подтверждаю
BEL, 07	Bell, звонок
BS, 08	Backspace, возврат на один символ назад
TAB, 09	Tab, горизонтальная табуляция
LF, 0A	Line Feed, перевод строки. Сейчас в большинстве языков программирования обозначается как \n
VT, 0B	Vertical Tab, вертикальная табуляция
FF, 0C	Form Feed, прогон страницы, новая страница
CR, 0D	Carriage Return, возврат каретки. Сейчас в большинстве языков программирования обозначается как \r
SO, 0E	Shift Out, изменить цвет красящей ленты в печатающем устройстве
SI, 0F	Shift In, вернуть цвет красящей ленты в печатающем устройстве обратно
DLE, 10	Data Link Escape, переключение канала на передачу данных
DC1, 11 DC2, 12 DC3, 13 DC4, 14	Device Control, символы управления устройствами
NAK, 15	Negative Acknowledgment, не подтверждаю
SYN, 16	Synchronization, символ синхронизации
ETB, 17	End of Text Block, конец текстового блока
CAN, 18	Cancel, отмена переданного ранее
EM, 19	End of Medium, конец носителя данных
SUB, 1A	Substitute, подставить. Ставится на месте символа, значение которого было потеряно или испорчено при передаче
ESC, 1B	Escape, управляющая последовательность
FS, 1C	File Separator, разделитель файлов
GS, 1D	Group Separator, разделитель групп
RS, 1E	Record Separator, разделитель записей
US, 1F	Unit Separator, разделитель юнитов
DEL, 7F	Delete, стереть последний символ

ПРИЛОЖЕНИЕ В

Функции для работы со строковыми переменными

addcslashes – экранирует строку слешами в стиле языка C.

addslashes – экранирует строку с помощью слешей.

bin2hex – преобразует бинарные данные в шестнадцатеричное представление.

chop – псевдоним rtrim.

chr – генерирует односимвольную строку по заданному числу.

chunk_split – разбивает строку на фрагменты.

convert_cyr_string – преобразует строку из одной кириллической кодировки в другую.

convert_uudecode – декодирует строку из формата uuencode в обычный вид.

convert_uuencode – кодирует строку в формат uuencode.

count_chars – возвращает информацию о символах, входящих в строку.

crc32 – вычисляет полином CRC32 для строки.

crypt – необратимое хеширование строки.

echo – выводит одну или более строк.

explode – разбивает строку с помощью разделителя.

fprintf – записывает отформатированную строку в поток.

get_html_translation_table – возвращает таблицу преобразований, используемую функциями htmlspecialchars и htmlentities.

hebrev – преобразует текст на иврите из логической кодировки в визуальную.

hebrevc – преобразует текст на иврите из логической кодировки в визуальную с преобразованием перевода строки.

hex2bin – преобразует шестнадцатеричные данные в двоичные.

html_entity_decode – преобразует HTML-сущности в соответствующие им символы.

htmlentities – преобразует все возможные символы в соответствующие HTML-сущности.

htmlspecialchars_decode – преобразует специальные HTML-сущности обратно в соответствующие символы.

htmlspecialchars – преобразует специальные символы в HTML-сущности.

implode – объединяет элементы массива в строку.

join – псевдоним implode.

lcfirst – преобразует первый символ строки в нижний регистр.

levenshtein – вычисляет расстояние Левенштейна между двумя строками.

localeconv – возвращает информацию о форматировании чисел.

ltrim – удаляет пробелы (или другие символы) из начала строки.

md5_file – возвращает MD5-хеш файла.

md5 – возвращает MD5-хеш строки.

metaphone – возвращает ключ metaphone для строки.

money_format – форматирует число как денежную величину.

nl_langinfo – возвращает информацию о языке и локали.

nl2br – вставляет html-код разрыва строки перед каждым переводом строки.

number_format – форматирует число с разделением групп.

ord – конвертирует первый байт строки в число от 0 до 255.

parse_str – разбирает строку в переменные.

print – выводит строку.

printf – выводит отформатированную строку.

quoted_printable_decode – преобразует строку, закодированную методом quoted-printable, в 8-битную строку.

quoted_printable_encode – преобразует 8-битную строку с помощью метода quoted-printable.

quotemeta – экранирует специальные символы.

rtrim – удаляет пробелы (или другие символы) из конца строки.

setlocale – устанавливает настройки локали.

sha1_file – возвращает SHA1-хеш файла.

sha1 – возвращает SHA1-хеш строки.

similar_text – вычисляет степень похожести двух строк.

soundex – возвращает ключ soundex для строки.

sprintf – возвращает отформатированную строку.

sscanf – разбирает строку в соответствии с заданным форматом.

str_getcsv – выполняет разбор CSV-строки в массив.

str_ireplace – регистронезависимый вариант функции str_replace.

str_pad – дополняет строку другой строкой до заданной длины.

str_repeat – возвращает повторяющуюся строку.

str_replace – заменяет все вхождения строки поиска на строку замены.

str_rot13 – выполняет преобразование ROT13 над строкой.

str_shuffle – переставляет символы в строке случайным образом.

str_split – преобразует строку в массив.

str_word_count – возвращает информацию о словах, входящих в строку.

strcasecmp – бинарно-безопасное сравнение строк без учета регистра.

strchr – псевдоним strstr.

strcmp – бинарно-безопасное сравнение строк.

strcoll – сравнение строк с учетом текущей локали.

strcspn – возвращает длину участка в начале строки, не соответствующего маске.

strip_tags – удаляет теги HTML и PHP из строки.

stripcslashes – удаляет экранирование символов, произведенное функцией addcslashes.

stripos – возвращает позицию первого вхождения подстроки без учета регистра.

stripslashes – удаляет экранирование символов.

stristr – регистронезависимый вариант функции strstr.

strlen – возвращает длину строки.

strnatcasecmp – сравнение строк без учета регистра с использованием алгоритма «natural order».

strnatcmp – сравнение строк с использованием алгоритма «natural order».

strncasecmp – бинарно-безопасное сравнение первых *n* символов строк без учета регистра.

strncmp – бинарно-безопасное сравнение первых *n* символов строк.

strpbrk – ищет в строке любой символ из заданного набора.

strpos – возвращает позицию первого вхождения подстроки.

strrchr – находит последнее вхождение символа в строке.

strrev – переворачивает строку задом наперед.

strripos – возвращает позицию последнего вхождения подстроки без учета регистра.

strrpos – возвращает позицию последнего вхождения подстроки в строке.

strspn – возвращает длину участка в начале строки, полностью соответствующего маске.

strstr – находит первое вхождение подстроки.

strtok – разбивает строку на токены.

strtolower – преобразует строку в нижний регистр.

strtoupper – преобразует строку в верхний регистр.

strtr – преобразует заданные символы или заменяет подстроки.

substr_compare – бинарно-безопасное сравнение двух строк со смещением, с учетом или без учета регистра.

substr_count – возвращает число вхождений подстроки.

substr_replace – заменяет часть строки.

substr – возвращает подстроку.

trim – удаляет пробелы (или другие символы) из начала и конца строки.

ucfirst – преобразует первый символ строки в верхний регистр.

ucwords – преобразует в верхний регистр первый символ каждого слова в строке.

vfprintf – записывает отформатированную строку в поток.

vprintf – выводит отформатированную строку.

vsprintf – возвращает отформатированную строку.

wordwrap – переносит строку по указанному количеству символов.

ПРИЛОЖЕНИЕ Г

Функции для работы с массивами

array_change_key_case – меняет регистр всех ключей в массиве.

array_chunk – разбивает массив на части.

array_column – возвращает массив из значений одного столбца входного массива.

array_combine – создает новый массив, используя один массив в качестве ключей, а другой для его значений.

array_count_values – подсчитывает количество всех значений массива.

array_diff_assoc – вычисляет расхождение массивов с дополнительной проверкой индекса.

array_diff_key – вычисляет расхождение массивов, сравнивая ключи.

array_diff_uassoc – вычисляет расхождение массивов с дополнительной проверкой индекса, осуществляемой при помощи callback-функции.

array_diff_ukey – вычисляет расхождение массивов, используя callback-функцию для сравнения ключей.

array_diff – вычисляет расхождение массивов.

array_fill_keys – создает массив и заполняет его значениями с определенными ключами.

array_fill – заполняет массив значениями.

array_filter – фильтрует элементы массива с помощью callback-функции.

array_flip – меняет местами ключи с их значениями в массиве.

array_intersect_assoc – вычисляет схождение массивов с дополнительной проверкой индекса.

array_intersect_key – вычисляет пересечение массивов, сравнивая ключи.

array_intersect_uassoc – вычисляет схождение массивов с дополнительной проверкой индекса, осуществляемой при помощи callback-функции.

array_intersect_ukey – вычисляет схождение массивов, используя callback-функцию для сравнения ключей.

array_intersect – вычисляет схождение массивов.

array_key_exists – проверяет, присутствует ли в массиве указанный ключ или индекс.

array_key_first – получает первый ключ массива.

array_key_last – получает последний ключ массива.

array_keys – возвращает все или некоторое подмножество ключей массива.

array_map – применяет callback-функцию ко всем элементам указанных массивов.

array_merge_recursive – рекурсивное слияние одного или более массивов.

array_merge – сливает один или большее количество массивов.

array_multisort – сортирует несколько массивов или многомерные массивы.

array_pad – дополняет массив определенным значением до указанной длины.

array_pop – извлекает последний элемент массива.

array_product – вычисляет произведение значений массива.

array_push – добавляет один или несколько элементов в конец массива.

array_rand – выбирает один или несколько случайных ключей из массива.

array_reduce – итеративно уменьшает массив к единственному значению с помощью callback-функции.

array_replace_recursive – рекурсивно заменяет элементы первого массива элементами переданных массивов.

array_replace – заменяет элементы массива элементами других переданных массивов.

array_reverse – возвращает массив с элементами в обратном порядке.

array_search – осуществляет поиск данного значения в массиве и возвращает ключ первого найденного элемента в случае удачи.

array_shift – извлекает первый элемент массива.

array_slice – выбирает срез массива.

array_splice – удаляет часть массива и заменяет ее чем-нибудь еще.

array_sum – вычисляет сумму значений массива.

array_udiff_assoc – вычисляет расхождение в массивах с дополнительной проверкой индексов, используя для сравнения значений callback-функцию.

array_udiff_uassoc – вычисляет расхождение в массивах с дополнительной проверкой индексов, используя для сравнения значений и индексов callback-функцию.

array_udiff – вычисляет расхождение массивов, используя для сравнения callback-функцию.

array_uintersect_assoc – вычисляет пересечение массивов с дополнительной проверкой индексов, используя для сравнения значений callback-функцию.

array_uintersect_uassoc – вычисляет пересечение массивов с дополнительной проверкой индекса, используя для сравнения индексов и значений индивидуальные callback-функции.

array_uintersect – вычисляет пересечение массивов, используя для сравнения значений callback-функцию.

array_unique – убирает повторяющиеся значения из массива.

array_unshift – добавляет один или несколько элементов в начало массива.

array_values – выбирает все значения массива.

array_walk_recursive – рекурсивно применяет пользовательскую функцию к каждому элементу массива.

array_walk – применяет заданную пользователем функцию к каждому элементу массива.

array – создает массив.

arsort – сортирует массив в обратном порядке, сохраняя ключи.

asort – сортирует массив, сохраняя ключи.

compact – создает массив, содержащий названия переменных и их значения.

count – подсчитывает количество элементов массива или чего-либо в объекте.

current – возвращает текущий элемент массива.

each – возвращает текущую пару ключ / значение из массива и смещает его указатель.

end – устанавливает внутренний указатель массива на его последний элемент.

extract – импортирует переменные из массива в текущую таблицу символов.

in_array – проверяет, присутствует ли в массиве значение.

key_exists – псевдоним array_key_exists.

key – выбирает ключ из массива.

krsort – сортирует массив по ключам в обратном порядке.

ksort – сортирует массив по ключам.

list – присваивает переменным из списка значения подобно массиву.

natcasesort – сортирует массив, используя алгоритм «natural order» без учета регистра символов.

natsort – сортирует массив, используя алгоритм «natural order».

next – перемещает указатель массива вперед на один элемент.

pos – псевдоним current.

prev – передвигает внутренний указатель массива на одну позицию назад.

range – создает массив, содержащий диапазон элементов.

reset – устанавливает внутренний указатель массива на его первый элемент.

rsort – сортирует массив в обратном порядке.

shuffle – перемешивает массив.

sizeof – псевдоним count.

sort – сортирует массив.

uasort – сортирует массив, используя пользовательскую функцию для сравнения элементов с сохранением ключей.

uksort – сортирует массив по ключам, используя пользовательскую функцию для сравнения ключей.

usort – сортирует массив по значениям используя пользовательскую функцию для сравнения элементов.

ЛИТЕРАТУРА

1. Скляр, Д. Изучаем PHP 7: руководство по созданию интерактивных web-сайтов.: пер. с англ. / Д. Скляр. − СПб.: ООО «Альфа-книга», 2017. − 464 с.

2. Котеров, Д. В. PHP 7 / Д. В. Котеров, И. В. Симлянов. − СПб.: БХВ-Петербург, 2016. − 1088 с.

3. Кузнецов, М. В. Самоучитель PHP 7 / М. В. Кузнецов, И. В. Симдянов. − СПб.: БХВ-Петербург, 2018. − 448 с.

4. Никсон, Р. Создаем динамические web-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. – 4-е изд. – СПб.: Питер, 2016. − 768 с.

5. Прохоренок, Н. HTML, JavaScript, PHP и MySQL. Джентльменский набор web-мастера / Н. Прохоренок, В. Дронов. – СПб.: БХВ-Петербург, 2015. – 766 с.

6. Зандстра, М. PHP. Объекты, шаблоны и методики программирования / М. Зандстра. – Вильямс, 2015. – 576 с.

ОГЛАВЛЕНИЕ

Учебное издание

Романенко Дмитрий Михайлович
Осоко Сергей Анатольевич

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PHP

Учебно-методическое пособие

Редактор *Е. И. Гоман*
Компьютерная верстка *А. А. Селиванова*
Дизайн обложки *П. П. Падалец*
Корректор *Е. И. Гоман*

Подписано в печать 02.04.2021. Формат $60\times84^{1}/_{16}$.
Бумага офсетная. Гарнитура Таймс. Печать ризографическая.
Усл. печ. л. 14,8. Уч.-изд. л. 15,3.
Тираж 80 экз. Заказ .

Издатель и полиграфическое исполнение:
УО «Белорусский государственный технологический университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/227 от 20.03.2014.
Ул. Свердлова, 13а, 220006, г. Минск.