*Лекция 3: Управляющие конструкции*

### **Условные операторы**

#### **Оператор if**

Это один из самых важных операторов многих языков, включая PHP. Он позволяет выполнять фрагменты кода в зависимости от условия. Структуру оператора *if* можно представить следующим образом:

if (выражение) блок\_выполнения

Здесь выражение есть любое правильное PHP-выражение (т.е. все, что имеет значение). В процессе обработки скрипта выражение преобразуется к *логическому типу*. Если в результате преобразования значение выражения истинно ( True ), то выполняется блок\_выполнения. В противном случае блок\_выполнения игнорируется. Если блок\_выполнения содержит несколько команд, то он должен быть заключен в фигурные скобки { }.

Правила преобразования выражения к *логическому типу*:

1. В FALSE преобразуются следующие значения:
   * логическое False
   * целый ноль ( 0 )
   * действительный ноль ( 0.0 )
   * пустая строка и строка "0"
   * массив без элементов
   * объект без переменных (подробно об объектах будет рассказано в одной из следующих лекций)
   * специальный тип NULL
2. Все остальные значения преобразуются в TRUE.

<?

$names = array("Иван","Петр","Семен");

if ($names[0]=="Иван") {

echo "Привет, Ваня!";

$num = 1;

$account = 2000;

}

if ($num) echo "Иван первый в списке!";

$bax = 30;

if ($account > 100\*$bax+3)

echo "Эта строчка не появится

на экране, так как условие не выполнено";

?>

Пример 3.1. Условный оператор if

#### **Оператор else**

Мы рассмотрели только одну, основную часть оператора *if* . Существует несколько расширений этого *оператора. Оператор* *else* расширяет *if* на случай, если проверяемое в *if* выражение является неверным, и позволяет выполнить какие-либо действия при таких условиях.

Структуру оператора *if* , расширенного с помощью оператора *else* , можно представить следующим образом:

if (выражение) блок\_выполнения

else блок\_выполнения1

Эту конструкцию if...else можно интерпретировать примерно так: если выполнено условие (т.е. выражение=true ), то выполняем действия из блока\_выполнения, иначе – действия из блока\_выполнения1. Использовать оператор *else* не обязательно.

Посмотрим, как можно изменить предыдущий пример, учитывая необходимость совершения действий и в случае невыполнения условия.

<?

$names = array("Иван","Петр","Семен");

if ($names[0]=="Иван") {

echo "Привет, Ваня!";

$num = 1;

$account = 2000;

} else {

echo "Привет, $names[0].

А мы ждали Ваню :(";

}

if ($num) echo "Иван первый в списке!";

else echo "Иван НЕ первый в списке?!";

$bax = 30;

if ($account > 100\*$bax+3)

echo "Эта строка не появится на экране,

так как условие не выполнено";

else echo "Зато появится эта строка!";

?>

Пример 3.2. Оператор else

#### **Оператор elseif**

Еще один способ расширения *условного оператора* *if* – использование оператора *elseif* . *elseif* – это комбинация *else* и *if* . Как и *else* , он расширяет *if* для выполнения различных действий в том случае, если условие, проверяемое в *if* , неверно. Но в отличие от *else* , альтернативные действия будут выполнены, только если *elseif*-условие является верным. Структуру оператора *if* , расширенного с помощью операторов *else* и *elseif* , можно представить следующим образом:

if (выражение) блок\_выполнения

elseif(выражение1) блок\_выполнения1

...

else блок\_выполненияN

Операторов *elseif* может быть сразу несколько в одном if-блоке. *Elseif*-утверждение будет выполнено, только если предшествующее if-условие является False, все предшествующие *elseif*-условия являются False, а данное *elseif*-условие – True.

<?

$names = array("Иван","Петр","Семен");

if ($names[0]=="Иван") {

// если первое имя в массиве Иван

echo "Привет, Ваня!";

}elseif ($names[0] == "Петр"){

// если первое имя

// не Иван, а Петр

echo "Привет, Петя!";

}elseif ($names[0] == "Семен"){

// если первое имя не

// Иван, не Петр, а Семен

echo "Привет, Сеня!";

}else {

// если первое имя не Иван,

// не Петр и не Семен

echo "Привет, $names[0]. А ты кто такой?";

}

?>

Пример 3.3. Оператор elseif

#### **Альтернативный синтаксис**

PHP предлагает *альтернативный синтаксис* для некоторых своих управляющих структур, а именно для *if* , *while* , *for* , *foreach*и *switch* . В каждом случае открывающую скобку нужно заменить на двоеточие ( :), а закрывающую – на endif;, endwhile; и т.д. соответственно.

Например, синтаксис оператора *if* можно записать таким образом:

if(выражение): блок\_выполнения endif;

Смысл остается тем же: если условие, записанное в круглых скобках оператора *if* , оказалось истиной, будет выполняться весь код, от двоеточия " :" до команды endif;. Использование такого синтаксиса полезно при встраивании php в html-код.

<?php

$names = array("Иван","Петр","Семен");

if ($names[0]=="Иван"):

?>

Привет, Ваня!

<?php

endif; ?>

Пример 3.4. Использование альтернативного синтаксиса

Если используются конструкции *else* и *elseif* , то также можно задействовать *альтернативный синтаксис*:

<?php

$a=1;

if ($a == 5):

print "a равно 5";

print "...";

elseif ($a == 6):

print "a равно 6";

print "!!!";

else:

print "a не равно ни 5, ни 6";

endif;

?>

#### **Оператор switch**

Еще одна конструкция, позволяющая проверять условие и выполнять в зависимости от этого различные действия, – это *switch* . На русский язык название данного оператора можно перевести как "переключатель". И смысл у него именно такой. В зависимости от того, какое значение имеет переменная, он переключается между различными блоками действия. *switch* очень похож на оператор if...*elseif*...else или набор операторов *if* . Структуру *switch* можно записать следующим образом:

switch (выражение или переменная){

case значение1:

блок\_действий1

break;

case значение2:

блок\_действий2

break;

...

default:

блок\_действий\_по\_умолчанию

}

В отличие от *if* , здесь значение выражения не приводится к логическому типу, а просто сравнивается со значениями, перечисленными после ключевых слов *case* ( значение1, значение2 и т.д.). Если значение выражения совпало с каким-то вариантом, то выполняется соответствующий блок\_действий – от двоеточия после совпавшего значения до конца *switch* или до первого оператора *break* , если таковой найдется. Если значение выражения не совпало ни с одним из вариантов, то выполняются действия по умолчанию ( блок\_действий\_по\_умолчанию ), находящиеся после ключевого слова default. Выражение в *switch*вычисляется только один раз, а в операторе *elseif* – каждый раз, поэтому, если выражение достаточно сложное, то *switch*работает быстрее.

Пример [3.3](https://www.intuit.ru/studies/courses/42/42/lecture/27179?page=1#example.3.3) можно переписать с использованием *switch* следующим образом:

<?

$names = array("Иван","Петр","Семен");

switch ($names[0]){

case "Иван":

echo "Привет, Ваня!";

break;

case "Петр":

echo "Привет, Петя!";

break;

case "Семен":

echo "Привет, Сеня!";

break;

default:

echo "Привет, $names[0].

А как Вас зовут?";

}

?>

Если в этом примере опустить оператор *break* , например, в *case* "Петр":, то, если переменная окажется равной строке "Петр", после вывода на экран сообщения "Привет, Петя!" программа пойдет дальше и выведет также сообщение "Привет, Сеня!" и только потом, встретив *break* , продолжит свое выполнение за пределами *switch* .

Для конструкции *switch* , как и для *if* , возможен *альтернативный синтаксис*, где открывающая *switch* фигурная скобка заменяется двоеточием, а закрывающая – endswitch; соответственно.

### **Циклы**

В *PHP* существует несколько конструкций, позволяющих выполнять повторяющиеся действия в зависимости от условия. Это циклы *while* , *do..while* , *foreach* и *for* . Рассмотрим их более подробно.

#### while

Структура:

while (выражение) { блок\_выполнения }

либо

while (выражение): блок\_выполнения endwhile;

*while* – простой цикл. Он предписывает PHP выполнять команды блока\_выполнения до тех пор, пока выражение вычисляется как True (здесь, как и в *if* , происходит приведение выражения к логическому типу). Значение выражения проверяется каждый раз в начале цикла, так что, даже если его значение изменилось в процессе выполнения блока\_выполнения, цикл не будет остановлен до конца итерации (т.е. пока все команды блока\_выполнения не будут исполнены).

<?

//эта программа напечатает все четные цифры

$i = 1;

while ($i < 10) {

if ($i % 2 == 0) print $i;

// печатаем цифру, если она четная

$i++;

// и увеличиваем $i на единицу

}

?>

Пример 3.5. Оператор while

#### **do... while**

Циклы *do..while* очень похожи на циклы *while* , с той лишь разницей, что истинность выражения проверяется в конце цикла, а не в начале. Благодаря этому блок\_выполнения цикла do...while гарантированно выполняется хотя бы один раз.

Структура:

do {блок\_выполнения} while (выражение);

<?

// эта программа напечатает число 12, несмотря на то

// что условие цикла не выполнено

$i = 12;

do{

if ($i % 2 == 0) print $i;

// если число четное, то печатаем его

$i++;

// увеличиваем число на единицу

}while ($i<10)

?>

Пример 3.6. Оператор do..while

#### **for**

Это самые сложные циклы в PHP. Они напоминают соответствующие циклы C.

Структура:

for (выражение1; выражение2; выражение3) {блок\_выполнения}

либо

for (выражение1; выражение2; выражение3): блок\_выполнения endfor;

Здесь, как мы видим, условие состоит сразу из трех выражений. Первое выражение выражение1 вычисляется безусловно один раз в начале цикла. В начале каждой итерации вычисляется выражение2. Если оно является True, то цикл продолжается и выполняются все команды блока\_выполнения. Если выражение2 вычисляется как False, то исполнение цикла останавливается. В конце каждой итерации (т.е. после выполнения всех команд блока\_выполнения ) вычисляется выражение3.

Каждое из выражений 1, 2, 3 может быть пустым. Если выражение2 является пустым, то это значит, что цикл должен выполняться неопределенное время (в этом случае PHP считает это выражение всегда истинным). Это не так бесполезно, как кажется, ведь цикл можно останавливать, используя оператор *break* .

Например, все четные цифры можно вывести с использованием цикла *for* таким образом:

<?php

for ($i=0; $i<10; $i++){

if ($i % 2 == 0) print $i;

// печатаем четные числа

}

?>

Если опустить второе выражение (условие $i<10 ), то такую же задачу можно решить, останавливая цикл оператором *break* .

<?php

for ($i=0; ; $i++){

if ($i>=10) break;

// если $i больше или равно 10,

// то прекращаем работу цикла

if ($i % 2 == 0) print $i;

// если число четное,

// то печатаем его

}

?>

Можно опустить все три выражения. В этом случае просто не будет задано начальное значение счетчика $i и оно не будет изменяться каждый раз в конце цикла. Все эти действия можно записать в виде отдельных команд либо в блоке\_выполнения, либо перед циклом:

<?php

$i=2; // задаем начальное значение счетчика

for ( ; ; ){

if ($i>=10) break;

// если $i больше или равно 10,

// то прекращаем работу цикла

if ($i % 2 == 0) print $i;

// если число четное,

// то печатаем его

$i++; // увеличиваем счетчик на единицу

}

?>

В третье выражение конструкции *for* можно записывать через запятую сразу несколько простейших команд. Например, если мы хотим просто вывести все цифры, то программу можно записать совсем просто:

<?php

for ($i=0; $i<10; print $i, $i++)

/\* Если блок\_выполнения не содержит команд

или содержит только одну команду,

фигурные скобки, в которые он заключен,

можно опускать\*/

?>

#### **foreach**

Еще одна полезная конструкция. Она появилась только в *PHP4* и предназначена исключительно для работы с массивами.

Синтаксис:

foreach ($array as $value) {блок\_выполнения}

либо

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

{блок\_выполнения}

В первом случае формируется цикл по всем элементам массива, заданного переменной $array. На каждом шаге цикла значение текущего элемента массива записывается в переменную $value, и внутренний счетчик массива передвигается на единицу (так что на следующем шаге будет виден следующий элемент массива). Внутри блока\_выполнения значение текущего элемента массива может быть получено с помощью переменной $value. Выполнение блока\_выполнения происходит столько раз, сколько элементов в массиве $array.

Вторая форма записи в дополнение к перечисленному выше на каждом шаге цикла записывает ключ текущего элемента массива в переменную $key, которую тоже можно использовать в блоке\_выполнения.

Когда *foreach* начинает исполнение, внутренний указатель массива автоматически устанавливается на первый элемент.

<?php

$names = array("Иван","Петр","Семен");

foreach ($names as $val) {

echo "Привет, $val <br>";

// выведет всем приветствие

}

foreach ($names as $k => $val) {

// кроме приветствия,

// выведем номера в списке, т.е. ключи

echo "Привет, $val !

Ты в списке под номером $k <br>";

}

?>

Пример 3.7. Оператор foreach

### **Операторы передачи управления**

Иногда требуется немедленно завершить работу *цикла* либо отдельной его итерации. Для этого используют *операторы* *break* и *continue* .

#### **Break**

Оператор *break* заканчивает выполнение текущего цикла, будь то *for* , *foreach* , *while* , *do..while* или *switch* . *break*может использоваться с числовым аргументом, который говорит, работу скольких управляющих структур, содержащих его, нужно завершить.

<?php

$i=1;

while ($i) {

$n = rand(1,10);

// генерируем произвольное число

// от 1 до 10

echo "$i:$n ";

// выводим номер итерации и

// сгенерированное число

if ($n==5) break;

/\* Если было сгенерировано число 5,

то прекращаем работу цикла. В этом случае

все, что находится после этой строчки

внутри цикла, не будет выполнено \*/

echo "Цикл работает <br>";

$i++;

}

echo "<br>Число итераций цикла $i ";

?>

Пример 3.8. Оператор break

Результатом работы этого скрипта будет примерно следующее:

1:7 Цикл работает

2:2 Цикл работает

3:5

Число итераций цикла 3

Если после оператора *break* указать число, то прервется именно такое количество содержащих этот *оператор циклов*. В приведенном выше примере это неактуально, поскольку *вложенных циклов* нет. Немного изменим наш скрипт:

<?php

$i=1;

while ($i) {

$n = rand(1,10);

// генерируем произвольное число

// от 1 до 10

switch ($n){

case 5:

echo "<font color=blue>

Выход из switch (n=$n)</font>";

break 1;

// прекращаем работу switch

// (первого содержащего break цикла)

case 10:

echo "<font color=red>

Выход из switch и

while (n=$n)</font>";

break 2;

// прекращаем работу switch и while

// (двух содержащих break циклов)

default:

echo "switch работает (n=$n), ";

}

echo " while работает – шаг $i <br>";

$i++;

}

echo "<br>Число итераций цикла $i ";

?>

#### **continue**

Иногда нужно не полностью прекратить работу цикла, а только начать его новую итерацию. Оператор *continue* позволяет пропустить дальнейшие инструкции из блока\_выполнения любого цикла и продолжить выполнение с нового круга. *continue* можно использовать с числовым аргументом, который указывает, сколько содержащих его *управляющих конструкций* должны завершить работу.

Заменим в примере предыдущего параграфа оператор *break* на *continue* . Кроме того, ограничим число шагов цикла четырьмя.

<?php

$i=1;

while ($i<=4) {

$n = rand(1,10);

// генерируем произвольное число

// от 1 до 10

echo "$i:$n ";

// выводим номер итерации и

// сгенерированное число

if ($n==5) {

echo "Новая итерация <br>";

continue;

/\* Если было сгенерировано число 5,

то начинаем новую итерацию цикла,

$i не увеличивается \*/

}

echo "Цикл работает <br>";

$i++;

}

--$i;

echo "<br>Число итераций цикла $i ";

?>

Результатом работы этого скрипта будет

1:10 Цикл работает

2:5 Новая итерация

2:1 Цикл работает

3:1 Цикл работает

4:1 Цикл работает

Число итераций цикла 4

Заметим, что после выполнения оператора *continue* работа цикла не заканчивается. В примере счетчик цикла не меняется в случае получения числа 5, поскольку он находится после оператора *continue* . Фактически с помощью *continue* мы пытаемся избежать ситуации, когда будет сгенерировано число 5. Поэтому можно было просто написать, заменив оператор *continue* на проверку истинности выражения:

<?php

$i=1;

while ($i<4) {

$n = rand(1,10);

// генерируем произвольное число

// от 1 до 10

if ($n!==5) {

echo "$i:$n <br>";

// выводим номер итерации

// и сгенерированное число

$i++;

}

}

?>

В PHP существует одна особенность использования оператора *continue* – в конструкциях *switch* он работает так же, как и *break*. Если *switch* находится внутри цикла и нужно начать новую итерацию цикла, следует использовать continue 2.

### **Операторы включения**

#### **include**

Оператор *include* позволяет включать код, содержащийся в указанном файле, и выполнять его столько раз, сколько программа встречает этот *оператор*. Включение может производиться любым из перечисленных способов:

include 'имя\_файла';

include $file\_name;

include ("имя\_файла");

**Пример 3.9**. Пусть в файле params.inc у нас хранится набор каких-то параметров и функций. Каждый раз, когда нам нужно будет использовать эти параметры (функции), мы будем вставлять в текст нашей основной программы команду include ('params.inc').

params.inc

<?php

$user = "Вася";

$today = date("d.m.y");

/\* функция date() возвращает дату

и время (здесь – дату в формате

день.месяц.год) \*/

?>

include.php

<?php

include ("params.inc");

/\* переменные $user и $today заданы в файле

params.inc. Здесь мы тоже можем ими

пользоваться благодаря команде

include("params.inc") \*/

echo "Привет, $user!<br>";

// выведет "Привет, Вася!"

echo "Сегодня $today";

// выведет, например, "Сегодня 7.07.05"

?>

Пример 3.9. Использование оператора включения include

Заметим, что использование оператора *include* эквивалентно простой вставке содержательной части файла params.inc в код программы include.php. Может быть, тогда можно было в params.inc записать простой текст без всяких тегов, указывающих на то, что это php-код? Нельзя! Дело в том, что в момент вставки файла происходит переключение из режима обработки PHP в режим HTML. Поэтому код внутри включаемого файла, который нужно обработать как *PHP-скрипт*, должен быть заключен в соответствующие теги.

Поиск файла для вставки происходит по следующим правилам.

1. Сначала ведется поиск файла в *include\_path* относительно текущей рабочей директории.
2. Если файл не найден, то поиск производится в *include\_path* относительно директории текущего скрипта.
3. Параметр *include\_path* , определяемый в *файле настроек* PHP, задает имена директорий, в которых нужно искать *включаемые файлы*.

Например, ваш *include\_path* это . (то есть *текущая директория*), текущая рабочая директория это /www/. В основной файл include.php вы включаете файл my\_dir/a.php, который в свою очередь включает b.php. Тогда *парсер* первым делом ищет файл b.php в директории /www/, и если такового нет, то в директории /www/my\_dir/.

Если файл включен с помощью *include* , то содержащийся в нем код наследует область видимости переменных строки, где появился *include* . Любые переменные вызванного файла будут доступны в вызывающем файле с этой строки и далее. Соответственно, если *include* появляется внутри функции вызывающего файла, то код, содержащийся в вызываемом файле, будет вести себя так, как будто он был определен внутри функции. Таким образом, он унаследует область видимости этой функции. Хотя мы и не знакомились еще с понятием функции, все же приводим здесь эти сведения в расчете на интуитивное его понимание.

**Пример 3.10**. Пусть файл для вставки params.inc останется таким же, а include.php будет следующим:

<?php

function Footer(){

// объявляем функцию с именем Footer

include ("params.inc");

/\* включаем файл params.inc.

Теперь его переменными можно пользоваться,

но только внутри функции \*/

$str = "Сегодня: $today <br>";

$str .= "<a

href='mailto:help@intuit.ru'>Страницу

создал $user</a>";

echo "$str";

}

Footer();

// вызываем функцию Footer(). Получим:

//Сегодня: 08.07.05

//Страницу создал Вася

echo "$user, $today";

// выведет запятую, так как

// эти переменные видны только

// внутри функции

?>

Пример 3.10. Область видимости при использовани include

Кроме локальных файлов, с помощью *include* можно включать и внешние файлы, указывая их url-адреса. Данная возможность контролируется директивой url\_fopen\_wrappers в *файле настроек* PHP и по умолчанию, как правило, включена. Но в версиях PHP для Windows до PHP 4.3.0 эта возможность не поддерживается совсем, вне зависимости от url\_fopen\_wrappers.

include() – это специальная языковая конструкция, поэтому при использовании внутри условных блоков ее нужно заключать в фигурные скобки.

<?php

/\* Это неверная запись. Получим ошибку.

Мы же вставляем не одну команду,

а несколько, они только записаны

в другом файле \*/

if ($condition) include("first.php");

else include("second.php");

// А вот так правильно.

if ($condition){ include("first.php"); }

else { include("second.php"); }

?>

Пример 3.11. Использование include()

При использовании *include* возможны два вида ошибок – ошибка вставки (например, нельзя найти указанный файл, неверно написана сама команда вставки и т.п.) или ошибка исполнения (если ошибка содержится во вставляемом файле). В любом случае при ошибке в команде *include* исполнение скрипта не завершается.

#### **require**

Этот оператор действует примерно так же, как и #include в C++. Все, что мы говорили о *include* , лишь за некоторыми исключениями, справедливо и для *require* . *require* также позволяет включать в программу и исполнять какой-либо файл. Основное отличие *require* и *include* заключается в том, как они реагируют на возникновение ошибки. Как уже говорилось, *include* выдает предупреждение, и работа скрипта продолжается. Ошибка в *require* вызывает фатальную ошибку работы скрипта и прекращает его выполнение.

*Условные операторы* на require() не влияют. Хотя, если строка, в которой появляется этот оператор, не исполняется, то ни одна строка кода из вставляемого файла тоже не исполняется. Циклы также не влияют на require(). Хотя код, содержащийся во вставляемом файле, является объектом цикла, но вставка сама по себе происходит только однажды.

В реализациях PHP до версии 4.0.2 использование require() означало, что интерпретатор обязательно попытается прочесть вставляемый файл.

*require* , как и *include* , при использовании внутри условных блоков нужно заключать в фигурные скобки.

### **Решение задачи**

И наконец, вернемся к задаче, сформулированной в начале лекции. Мы хотим создать программу, которую можно было бы использовать для отправки писем (или просто для их генерации) с приглашениями на различные мероприятия множеству пользователей. В предыдущей лекции уже рассматривался подобный случай. Сейчас мы вынесем всю информацию о людях и событиях в отдельный *файл* data.php и напишем программу, не зависящую (ну, может, совсем чуть-чуть зависящую) от этой информации и ее структуры. В этом случае для того, чтобы, например, расширить *список* адресатов, не нужно будет изменять *скрипт*, генерирующий приглашения. Кроме того, можно будет использовать информацию о людях и событиях в других скриптах. В самом скрипте, генерирующем приглашения letters.php, мы использовали *условные операторы*, циклы, *require* и другие изученные ранее конструкции.

<?php

define("SIGN","С уважением, Вася");

// пусть наша подпись

// будет константой

// информация о событиях

$events = array(

"f" => "день открытых дверей",

"o" => "открытие выставки",

"p" => "бал выпускников");

// имеющаяся информация о людях

// (имя и электронный адрес)

$people = array(

"ivan" => array(

"name" => "Иван Иванович",

"email"=>"user\_ivan@intuit.ru"),

"pit" => array(

"name" => "Петр Петрович",

"email" => "user\_petr@intuit.ru"),

"semen" => array(

"name" => "Семен Семенович"));

// кто куда приглашается

$who\_where["ivan"] = "o" ;

// Иван – на выставку

$who\_where["pit"] = "p";

// Петр – на бал

$who\_where["semen"] = "f";

// Семен – на день открытых дверей

?>

Листинг 3.12. data.php

<?php

require("data.php");

// включаем файл с данными о событиях

foreach($people as $key => $man\_info){

// для каждого человека делаем следующее:

$event\_key = $who\_where[$key];

// получаем событие,

// на которое он приглашается

if ($event\_key<>""){

foreach($man\_info as $key1 => $info){

// получаем имя и email

// конкретного человека

if ($key1=="name")

$str = "Уважаемый (ая), $info";

if ($key1=="email") $email = $info;

}

// составляем приглашение

$str .= "<br>Приглашаем Вас на ".

$events[$event\_key];

switch ($event\_key){

// в зависимости от события

// добавляем какую-нибудь строчку

case "f":

$str .= "<br>Подтвердите Ваше

участие по телефону!";

break;

case "o":

$str .= "<br>Приходите за 15

минут до открытия!";

break;

case "p":

$str .= "<br>Не забудьте подарок :-)";

break;

}

$str .= "<br>" . SIGN . "<hr>";

// добавляем подпись

echo $str; // выводим приглашение на экран

/\* если у вас настроена отправка почты с

помощью PHP, то письмо можно отправить

командой mail($email,"Letter",$str); \*/

}

}

?>

Листинг 3.13. letters.php

### **Заключение**

Итак, мы завершили изучение *управляющих конструкций* языка *PHP*. Мы старались привести самое основное и не углубляться в различные нюансы тех или иных конструкций. Подробную информацию можно найти в руководстве по *PHP*.