*Лекция 5: Функции в PHP*

#### **Функции, определяемые пользователем**

Для чего нужны функции? Чтобы ответить на этот вопрос, нужно понять, что вообще представляют собой функции. В программировании, как и в математике, функция есть *отображение множества* ее аргументов на множество ее значений. То есть функция для каждого набора значений аргумента возвращает какие-то значения, являющиеся результатом ее работы. Зачем нужны функции, попытаемся объяснить на примере. Классический пример функции в программировании – это функция, вычисляющая значение факториала числа. То есть мы задаем ей число, а она возвращает нам его факториал. При этом не нужно для каждого числа, факториал которого мы хотим получить, повторять один и тот же код – достаточно просто вызвать функцию с аргументом, равным этому числу.

Функция вычисления факториала натурального числа

<?php

function fact($n){

if ($n==0) return 1;

else return $fact = $n \* fact($n-1);

}

echo fact(3);

// можно было бы написать echo (3\*2);

// но если число большое,

echo fact(50);

// то удобнее пользоваться функцией,

// чем писать echo (50\*49\*48\*...\*3\*2);

?>

Таким образом, когда мы осуществляем действия, в которых прослеживается зависимость от каких-то данных, и при этом, возможно, нам понадобится выполнять такие же действия, но с другими исходными данными, удобно использовать механизм функций – оформить блок действий в виде *тела функции*, а меняющиеся данные – в качестве ее параметров.

Посмотрим, как в общем виде выглядит задание (объявление) функции. Функция может быть определена с помощью следующего синтаксиса:

function Имя\_функции (параметр1, параметр2,

... параметрN){

Блок\_действий

return "значение, возвращаемое функцией";

}

Если прямо так написать в php-программе, то работать ничего не будет. Во-первых, Имя\_функции и имена *параметров функции* ( параметр1, параметр2 и т.д.) должны соответствовать правилам наименования в PHP (и русских символов в них лучше не использовать). Имена функций нечувствительны к регистру. Во-вторых, *параметры функции* – это переменные языка, поэтому перед названием каждого из них должен стоять знак $. Никаких многоточий ставить в списке параметров нельзя. В-третьих, вместо слов блок\_действий в *теле функции* должен находиться любой правильный PHP-код (не обязательно зависящий от параметров). И наконец, после ключевого слова *return* должно идти корректное php-выражение (что-либо, что имеет значение). Кроме того, у функции может и не быть параметров, как и возвращаемого значения. Пример правильного *объявления функции* – функция вычисления факториала, приведенная выше.

Как происходит вызов функции? Указывается имя функции и в круглых скобках список значений ее параметров, если таковые имеются:

<?php

Имя\_функции ("значение\_для\_параметра1",

"значение\_для\_параметра2",...);

// пример вызова функции – вызов функции

// вычисления факториала приведен выше,

// там для вычисления факториала числа 3

// мы писали: fact(3);

// где fact – имя вызываемой функции,

// а 3 – значение ее параметра с именем $n

?>

Когда можно вызывать функцию? Казалось бы, странный вопрос. Функцию можно вызвать после ее определения, т.е. в любой строке программы ниже блока function f\_name(){...}. В *PHP3* это было действительно так. Но уже в *PHP4* такого требования нет. Все дело в том, как интерпретатор обрабатывает получаемый код. Единственное исключение составляют функции, определяемые условно (внутри условных операторов или других функций). Когда функция определяется таким образом, ее определение должно предшествовать ее вызову.

<?php

$make = true;

/\* здесь нельзя вызвать Make\_event();

потому что она еще не существует, но можно

вызвать Save\_info() \*/

Save\_info("Вася","Иванов",

"Я выбрал курс по PHP");

if ($make){

// определение функции Make\_event()

function Make\_event(){

echo "<p>Хочу изучать Python<br>";

}

}

// теперь можно вызывать Make\_event()

Make\_event();

// определение функции Save\_info

function Save\_info($first, $last, $message){

echo "<br>$message<br>";

echo "Имя: ". $first . " ". $last . "<br>";

}

Save\_info("Федя","Федоров",

"А я выбрал Lisp");

// Save\_info можно вызывать и здесь

?>

Пример 5.1. Определение функции внутри условного оператора

Если функция однажды определена в программе, то переопределить или удалить ее позже нельзя. Несмотря на то, что имена функций нечувствительны к регистру, лучше вызывать функцию по тому же имени, каким она была задана в определении.

<?php

/\* нельзя сохранить данные, т.е. вызвать

функцию DataSave() до того, как выполнена

проверка их правильности, т.е. вызвана

функция DataCheck() \*/

DataCheck();

DataSave();

function DataCheck(){

// проверка правильности данных

function DataSave(){

// сохраняем данные

}

}

?>

Пример 5.2. Определение функции внутри функции

Рассмотрим подробнее аргументы функций, их назначение и использование.

#### **Аргументы функций**

У каждой функции может быть, как мы уже говорили, *список аргументов*. С помощью этих аргументов в функцию передается различная информация (например, значение числа, факториал которого надо подсчитать). Каждый аргумент представляет собой переменную или константу.

С помощью аргументов данные в функцию можно передавать тремя различными способами. Это передача аргументов по значению (используется по умолчанию), по ссылке и *задание значения аргументов по умолчанию*. Рассмотрим эти способы подробнее.

Когда аргумент передается в функцию по значению, изменение значения аргумента внутри функции не влияет на его значение вне функции. Чтобы позволить функции изменять ее аргументы, их нужно передавать по ссылке. Для этого в определении функции перед именем аргумента следует написать знак амперсанд "&".

<?php

// напишем функцию, которая бы добавляла

// к строке слово checked

function add\_label(&$data\_str){

$data\_str .= "checked";

}

$str = "<input type=radio name=article ";

// пусть имеется такая строка

echo $str ."<br>";

// выведет элемент формы –

// не отмеченную радио кнопку

add\_label($str);

// вызовем функцию

echo $str ."><br>";

// это выведет уже отмеченную

// радио кнопку

?>

Пример 5.3. Передача аргументов по ссылке

В функции можно определять значения аргументов, используемые по умолчанию. Само значение по умолчанию должно быть константным выражением, а не переменной и не представителем класса или вызовом другой функции.

У нас есть функция, создающая *информационное сообщение*, подпись к которому меняется в зависимости от значения переданного ей параметра. Если значение параметра не задано, то используется подпись "Оргкомитет.".

<?php

function Message($sign="Оргкомитет."){

// здесь параметр sign имеет по умолчанию значение "Оргкомитет"

echo "Следующее собрание состоится завтра.<br>";

echo $sign . "<br>";

}

Message();

// вызываем функцию без параметра.

// В этом случае подпись – это Оргкомитет

Message("С уважением, Вася.");

// В этом случае подпись

// будет "С уважением, Вася."

?>

Пример 5.4. Значения аргументов по умолчанию

Результатом работы этого скрипта будет:

Следующее собрание состоится завтра.

Оргкомитет.

Следующее собрание состоится завтра.

С уважением, Вася.

Если у функции несколько параметров, то те аргументы, для которых задаются значения по умолчанию, должны быть записаны после всех остальных аргументов в определении функции. В противном случае появится ошибка, если эти аргументы будут опущены при вызове функции.

Например, мы хотим внести описание статьи в каталог. Пользователь должен ввести такие характеристики статьи, как ее название, автора и краткое описание. Если пользователь не вводит имя автора статьи, считаем, что это Иванов Иван.

<?php

function Add\_article($title, $description,

$author="Иванов Иван"){

echo "Заносим в каталог статью: $title,";

echo "автор $author";

echo "<br>Краткое описание: ";

echo "$description <br>";

}

Add\_article("Информатика и мы",

"Это статья про информатику ...",

"Петров Петр");

Add\_article("Кто такие хакеры",

"Это статья про хакеров ...");

?>

В результате работы скрипта получим следующее

Заносим в каталог статью: Информатика и мы,

автор Петров Петр.

Краткое описание:

Это статья про информатику...

Заносим в каталог статью: Кто такие хакеры,

автор Иванов Иван.

Краткое описание:

Это статья про хакеров...

Если же мы напишем вот так:

<?php

function Add\_article($author="Иванов Иван",

$title, $description){

// ...действия как в предыдущем примере

}

Add\_article("Кто такие хакеры",

"Это статья про хакеров...");

?>

То в результате получим:

Warning: Missing argument 3 for

add\_article() in

c:\users\nina\tasks\func\def\_bad.php

on line 2

##### Списки аргументов переменной длины

В *PHP4* можно создавать функции с переменным числом аргументов. То есть мы создаем функцию, не зная заранее, со сколькими аргументами ее вызовут. Для написания такой функции никакого специального синтаксиса не требуется. Все делается с помощью *встроенных функций* *func\_num\_args()*, *func\_get\_arg()*, *func\_get\_args()*.

Функция ***func\_num\_args()*** возвращает число аргументов, переданных в текущую функцию. Эта функция может использоваться только внутри определения пользовательской функции. Если она появится вне функции, то интерпретатор выдаст предупреждение.

<?php

function DataCheck(){

$n = func\_num\_args();

echo "Число аргументов функции $n";

}

DataCheck();

// выведет строку

// "Число аргументов функции 0"

DataCheck(1,2,3);

// выведет строку

// "Число аргументов функции 3"

?>

Пример 5.5. Использование функции func\_num\_args()

Функция ***func\_get\_arg*** (целое номер\_аргумента ) возвращает аргумент из списка переданных в функцию аргументов, порядковый номер которого задан параметром номер\_аргумента. Аргументы функции считаются начиная с нуля. Как и *func\_num\_args()*, эта функция может использоваться только внутри определения какой-либо функции.

Номер\_аргумента не может превышать число аргументов, переданных в функцию. Иначе будет сгенерировано предупреждение, и функция *func\_get\_arg()* возвратит False.

Создадим функцию для проверки типа данных ее аргументов. Считаем, что проверка прошла успешно, если первый аргумент функции – целое число, второй – строка.

<?

function DataCheck(){

$check = true;

$n = func\_num\_args();

// число аргументов,

// переданных в функцию

/\* проверяем, является ли первый

переданный аргумент целым числом \*/

if ($n>=1)

if (!is\_int(func\_get\_arg(0)))

$check = false;

/\* проверяем, является ли второй

переданный аргумент строкой \*/

if ($n>=2)

if (!is\_string(func\_get\_arg(1)))

$check = false;

return $check;

}

if (DataCheck(123,"text"))

echo "Проверка прошла успешно<br>";

else echo "Данные не удовлетворяют

условиям<br>";

if (DataCheck(324))

echo "Проверка прошла успешно<br>";

else echo "Данные не удовлетворяют условиям<br>";

?>

Пример 5.6. Функция для проверки типа данных ее аргументов

Результатом работы будет следующее.

Проверка прошла успешно

Проверка прошла успешно

Функция ***func\_get\_args()*** возвращает массив, состоящий из списка аргументов, переданных функции. Каждый элемент массива соответствует аргументу, переданному функции. Если функция используется вне определения пользовательской функции, то генерируется предупреждение.

Перепишем предыдущий пример, используя эту функцию. Будем проверять, является ли целым числом каждый четный аргумент, передаваемый функции:

<?

function DataCheck(){

$check = true;

$n = func\_num\_args();

// число аргументов,

// переданных в функцию

$args = func\_get\_args();

// массив аргументов функции

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

$v = $args[$i];

if ($i % 2 == 0){

if (!is\_int($v)) $check = false;

// проверяем,

// является ли четный аргумент целым

}

}

return $check;

}

if (DataCheck("text", 324))

echo "Проверка прошла успешно<br>";

else echo "Данные не удовлетворяют

условиям<br>";

?>

Как видим, комбинации функций *func\_num\_args()*, *func\_get\_arg()* и *func\_get\_args()* используется для того, чтобы функции могли иметь переменный *список аргументов*. Эти функции были добавлены только в PHP 4. В *PHP3* для того, чтобы добиться подобного эффекта, можно использовать в качестве аргумента функции массив. Например, вот так можно написать скрипт, проверяющий, является ли каждый нечетный *параметр функции* целым числом:

<?

function DataCheck($params){

$check =true;

$n = count($params);

// число аргументов,

// переданных в функцию

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

$v = $params[$i];

if ($i % 2 != 0){

// проверяем, является ли нечетный

// аргумент целым

if (!is\_int($v)) $check = false;

}

}

return $check;

}

if (DataCheck("text", 324))

echo "Проверка прошла успешно<br>";

else echo "Данные не удовлетворяют условиям<br>";

?>

#### **Использование переменных внутри функции**

##### **Глобальные переменные**

Чтобы использовать внутри функции переменные, заданные вне ее, эти переменные нужно объявить как глобальные. Для этого в *теле функции* следует перечислить их имена после ключевого слова *global*:

global $var1, $var2;

<?

$a=1;

function Test\_g(){

global $a;

$a = $a\*2;

echo 'в результате работы функции $a=',$a;

}

echo 'вне функции $a=',$a,', ';

Test\_g();

echo "<br>";

echo 'вне функции $a=',$a,', ';

Test\_g();

?>

Пример 5.7. Глобальные переменные

В результате работы этого скрипта получим:

вне функции $a=1, в результате работы

функции $a=2

вне функции $a=2, в результате работы

функции $a=4

Когда переменная объявляется как глобальная, фактически создается ссылка на *глобальную переменную*. Поэтому такая запись эквивалентна следующей (массив $GLOBALS содержит все переменные, глобальные относительно текущей области видимости):

$var1 = &$GLOBALS["var1"];

$var2 = &$GLOBALS["var2"];

Это значит, например, что удаление переменной $var1 не удаляет *глобальной переменной* $GLOBALS["var1"].

##### **Статические переменные**

Чтобы использовать переменные только внутри функции, при этом сохраняя их значения и после выхода из функции, нужно объявить эти переменные как статические. ***Статические переменные*** видны только внутри функции и не теряют своего значения, если выполнение программы выходит за *пределы функции*. Объявление таких переменных производится с помощью ключевого слова *static*:

static $var1, $var2;

*Статической переменной* может быть присвоено любое значение, но не ссылка.

<?

function Test\_s(){

static $a = 1;

// нельзя присваивать выражение или ссылку

$a = $a\*2;

echo $a;

}

Test\_s(); // выведет 2

echo $a; // ничего не выведет, так как

// $a доступна только

// внутри функции

Test\_s(); // внутри функции $a=2, поэтому

// результатом работы функции

// будет число 4

?>

Пример 5.8. Использование статической переменной

#### **Возвращаемые значения**

Все функции, приведенные выше в качестве примеров, выполняли какие-либо действия. Кроме подобных действий, любая функция может возвращать как результат своей работы какое-нибудь значение. Это делается с помощью утверждения *return*. Возвращаемое значение может быть любого типа, включая списки и объекты. Когда интерпретатор встречает команду *return* в *теле функции*, он немедленно прекращает ее исполнение и переходит на ту строку, из которой была вызвана функция.

Например, составим функцию, которая возвращает возраст человека. Если человек не умер, то возраст считается относительно текущего года.

<?php

/\* если второй параметр вычисляется

как true, то он рассматривается как

дата смерти, \*/

function Age($birth, $is\_dead){

if ($is\_dead) return $is\_dead-$birth;

else return date("Y")-$birth;

}

echo Age(1971, false); // для 2009 года выведет 38

echo Age(1971, 2001); // выведет 30

?>

В этом примере можно было и не использовать функцию *return*, а просто заменить ее функцией вывода echo. Однако если мы все же делаем так, что функция возвращает какое-то значение (в данном случае возраст человека), то в программе мы можем присвоить любой переменной значение этой функции:

$an\_age = Age(1981, 2004);

В результате работы функции может быть возвращено только одно значение. Несколько значений можно получить, если возвращать список значений (*одномерный массив*). Допустим, мы хотим получить полный возраст человека с точностью до дня.

<?php

function Full\_age($b\_day, $b\_month, $b\_year)

{

$y = date("Y");

$m = intval(date("m"));

$d = intval(date("d"));

$b\_month = intval($b\_month);

$b\_day = intval($b\_day);

$b\_year = intval($b\_year);

$day = ($b\_day > $d ? 30 - $b\_day + $d : $d - $b\_day);

$tmpMonth = ($b\_day > $d ? -1 : 0);

$month = ($b\_month > $m + $tmpMonth ? 12 - $b\_month +

$tmpMonth + $m : $m+$tmpMonth - $b\_month);

$tmpYear = ($b\_month > $m + $tmpMonth ? -1 : 0);

if ($b\_year > $y + $tmpYear)

{

$year = 0; $month = 0; $day = 0;

}

else

{

$year = $y + $tmpYear - $b\_year;

}

return array ($day,$month,$year);

}

$age = Full\_age("29","06","1986");

echo "Вам $age[2] лет, $age[1] месяцев и $age[0] дней";

?>

Когда функция возвращает несколько значений для их обработки в программе, удобно использовать языковую конструкцию *list* (), которая позволяет одним действием присвоить значения сразу нескольким переменным. Например, в предыдущем примере, оставив без изменения функцию, обработать возвращаемые ей значения можно было так:

<?

// задание функции Full\_age()

list($day,$month,$year) = Full\_age("07",

"08","1974");

echo "Вам $year лет, $month месяцев и

$day дней";

?>

Вообще конструкцию *list* () можно использовать для присвоения переменным значений элементов любого массива.

<?

$arr = array("first","second");

list($a,$b) = $arr;

// переменной $a присваивается первое

// значение массива, $b – второе

echo $a," ",$b;

// выведет строку "first second"

?>

Пример 5.9. Использование list()

##### **Возвращение ссылки**

В результате своей работы функция также может возвращать ссылку на какую-либо переменную. Это может пригодиться, если требуется использовать функцию для того, чтобы определить, какой переменной должна быть присвоена ссылка. Чтобы получить из функции ссылку, нужно при объявлении перед ее именем написать знак амперсанд ( & ) и каждый раз при вызове функции перед ее именем тоже писать амперсанд ( & ). Обычно функция возвращает ссылку на какую-либо *глобальную переменную* (или ее часть – ссылку на элемент глобального массива), ссылку на *статическую переменную* (или ее часть) или ссылку на один из аргументов, если он был также передан по ссылке.

<?

$a = 3; $b = 2;

function & ref($par){

global $a, $b;

if ($par % 2 == 0) return $b;

else return $a;

}

$var =& ref(4);

echo $var, " и ", $b,"<br>";

//выведет 2 и 2

$b = 10;

echo $var, " и ", $b,"<br>";

// выведет 10 и 10

?>

Пример 5.10. Возвращение ссылки

При использовании синтаксиса ссылок в переменную $var нашего примера не копируется значение переменной $b возвращенной функцией $ref, а создается ссылка на эту переменную. То есть теперь переменные $var и $b идентичны и будут изменяться одновременно.

#### **Переменные функции**

PHP поддерживает концепцию *переменных функций*. Это значит, что если имя переменной заканчивается круглыми скобками, то PHP ищет функцию с таким же именем значения и пытается ее выполнить.

<?

/\* создадим две простые функции:

Add\_sign – добавляет подпись к строке и

Show\_text – выводит строку текста \*/

function Add\_sign($string,

$sign="С уважением, Петр"){

echo $string ." ".$sign;

}

function Show\_text(){

echo "Отправить сообщение по почте<br>";

}

$func = "Show\_text";

// создаем переменную со значением,

// равным имени функции Show\_text

$func();

// это вызовет функцию Show\_text

$func = "Add\_sign";

// создаем переменную со значением,

// равным имени функции Add\_sign

$func("Привет всем <br>");

// это вызовет функцию

// Add\_sign с параметром "Привет всем"

?>

Пример 5.11. Использование переменных функций

В этом примере функция Show\_text просто выводит строку текста. Казалось бы, зачем для этого создавать отдельную функцию, если существует специальная функция echo(). Дело в том, что такие функции, как echo(), print(), *unset*(), include() и т.п. нельзя использовать в качестве *переменных функций*. То есть если мы напишем:

<?

$func = "echo ";

$func("TEXT");

?>

то интерпретатор выведет ошибку:

Fatal error: Call to undefined function:

echo() in

c:\users\nina\tasks\func\var\_f.php on line 2

Поэтому для того, чтобы использовать любую из перечисленных выше функций как *переменную функцию*, нужно создать собственную функцию, что мы и сделали в предыдущем примере.

#### **Внутренние (встроенные) функции**

Говоря о функциях, определяемых пользователем, все же нельзя не сказать пару слов о *встроенных функциях*. С некоторыми из *встроенных функций*, такими как echo(), print(), date(), include(), мы уже познакомились. На самом деле все перечисленные функции, кроме date(), являются языковыми конструкциями. Они входят в ядро PHP и не требуют никаких дополнительных настроек и модулей. Функция date() тоже входит в состав ядра PHP и не требует настроек. Но есть и функции, для работы с которыми нужно установить различные библиотеки и подключить соответствующий модуль. Например, для использования функций работы с базой данных *MySql* следует скомпилировать PHP с поддержкой этого расширения. В последнее время наиболее распространенные расширения и соответственно их функции изначально включают в состав PHP так, чтобы с ними можно работать без каких бы то ни было дополнительных настроек интерпретатора.

#### **Решение задачи**

Напомним, в чем состоит задача. Мы хотим написать интерфейс, который позволял бы создавать *html-формы*. Пользователь выбирает, какие элементы и в каком количестве нужно создать, придумывает им названия, а наша программа сама генерирует требуемую форму.

Разобьем задачу на несколько подзадач: выбор типов элементов ввода и их количества, создание названий элементов ввода и обработка полученных данных, т.е. непосредственно генерация формы. Первая задача достаточно проста: нужно написать соответствующую форму, например подобную приведенной ниже ():

<form action="ask\_names.php">

Создать элемент "строка ввода текста": <input

type=checkbox name=types[]

value=string><br>

Количество элементов: <input type=text

name=numbers[string]

size=3><br>

<br>

Создать элемент "текстовая область": <input

type=checkbox

name=types[] value=text><br>

Количество элементов: <input type=text

name=numbers[text]

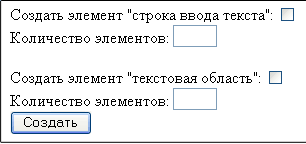
size=3><br>

<input type=submit value="Создать">

</form>

Листинг 5.12. task\_form.html

Когда мы пишем в имени элемента формы, например types[], это значит, что его имя – следующий элемент массива types. То есть у нас первый элемент формы ( "строка ввода текста" ) будет иметь имя types[0], а второй (текстовая область) – types[1]. В браузере task\_form.html будет выглядеть примерно так:



**Рис. 5.1.**Форма для выбора создаваемых элементов и их количества

После отправки данных этой формы мы получим информацию о том, какие элементы и сколько элементов каждого типа нужно создать. Следующий скрипт запрашивает названия для этих элементов:

<?

$file = "task.php";

/\* файл, который будет обрабатывать

сгенерированную этим скриптом форму \*/

function Ask\_names(){

// функция генерирует форму для

// ввода названий элементов ввода

global $file;

//объявляем, что хотим использовать эту

// переменную, заданную вне функции

if (isset($\_GET["types"])){

$st = '<form action="'.$file.'">';

foreach ($\_GET["types"] as $k => $type){

/\* перебираем все типы элементов,

которые нужно создать \*/

$num = $\_GET["numbers"][$type];

// сколько элементов каждого типа нужно

for ($i=1;$i<=$num;$i++){

// создаем $num строк для ввода

$st.= "Введите имя $i-го элемента типа $type: ";

$st.= "<input type=text name=names[$type][]><br>";

}

// сохраняем тип и число необходимых

// элементов ввода этого типа

$st.= "<input type=hidden name=types[] value=$type>";

$st.= "<input type=hidden name=numbers[] value=$num><br>";

}

$st .= "<input type=submit name=send value=send></form>";

return $st;

// в переменной $st содержится код формы

// для запроса имен

} else echo "Select type";

}

echo Ask\_names();

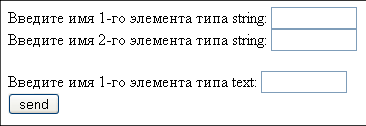
// вызываем функцию и выводим

// результаты ее работы

?>

Листинг 5.13. ask\_names.php

Допустим, нужно создать два элемента типа "текстовая строка" и один элемент типа "текстовая область", как и отмечено в форме выше. Тогда скрипт ask\_names.php обработает ее таким образом, что мы получим такую форму:



**Рис. 5.2.**Форма для ввода названий создаваемых элементов

Введем в эту форму, например, строки "Название", "Автор" и "Краткое содержание". Эти данные будет обрабатывать скрипт task.php.

<?

$show\_file = "task\_show.php";

/\* файл, который будет обрабатывать данные

созданной этим файлом формы \*/

function Create\_element($type,$name){

// функция создает элемент ввода

// по типу и названию

$str="";

switch($type){

case "string":

$str .= "$name: <input type=text name=string[]><br>";

break;

case "text":

$str .= "$name: <textarea name=text[]></textarea><br>";

break;

}

return $str;

}

function Create\_form(){

// функция создает форму

// с нужными элементами

global $show\_file;

$str = '<form action="'.$show\_file.'">';

foreach ($\_GET["types"] as $k => $type){

// перебираем типы элементов

$num = $\_GET["numbers"][$k];

// число элементов этого типа

for ($i=1;$i<=$num;$i++){

$arr = $\_GET["names"][$type][$i-1];

// имя создаваемого элемента

$str .= Create\_element($type,$arr);

// вызываем функцию для

// создания элемента

}

}

$str .= "<input type=submit value=send></form>";

echo $str;

}

$crt = "Create\_form";

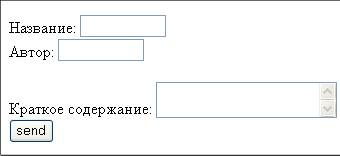
$crt(); // вызываем функцию создания

// формы Create\_form

?>

Листинг 5.14. task.php

Результатом работы этого скрипта с входными данными, приведенными выше, будет следующая форма:



**Рис. 5.3.**Пример формы, сгенерированной нашей программой

#### **Заключение**

Подведем итоги. В этой лекции мы изучили функции, определяемые пользователем, их синтаксис и семантику, способы передачи их аргументов и возвращаемых значений. Кроме того, обсуждались способы задания и работы с функциями, имеющими переменное число аргументов и альтернативный способ вызова функции (с помощью переменной, значение которой есть имя функции). В следующей лекции будет рассмотрена объектная модель языка PHP.