Вадим Дунаев

Основы PHP

Здесь рассматриваются основные понятия, синтаксис и классы (объекты) PHP без претензий на исчерпывающую полноту. Тем не менее, приведенный материал можно использовать в качестве краткого справочника и элементарного учебника для начинающих. Нумерация разделов согласована с моей книгой "**Самоучитель PHP**", ***3-е издание, 2007г.***.

2.1. Вывод данных

Начиная программировать на любом языке, важно уметь получать не только удручающие системные сообщения об ошибках, но и запланированные нами же сообщения, которые уведомляют, что программный код как-то понимается интерпретатором языка и даже выполняется. Поэтому важно сперва научиться выводить сообщения на экран монитора.

Вывод данных в окно Web-браузера с помощью PHP можно выполнить посредсвом оператора echo. Этот оператор позволяет вывести данные различных типов (числа, символьные строки и т.д.). Вы можете использовать его для вывода обычных текстовых строк, строк HTML-кода, а также данных других типов (чисел, массивов и др.) Посредством этого опеатора можно выводить в окно браузера как сообщения, предусмотренные вашим приложением, так и отладочные (значения переменных, возвращаемые значения выражений).

Синтаксис операторa вывода echo:

echo *элемент1, элемент2, ... , элементN*

Данные *элемент1, элемент2, ... , элементN*, подлежащие выводу, могут быть различных типов (см. разд. 2.3). Например, это могут быть числа, строки различных символов и др. Все данные, перечисленные в списке, выводятся в окне браузера друг за другом, без пробеов и запятых (рис. 2.1). Заметим, что результат в окне браузера будет таким же, как если бы мы применили оператор

```
echo "AAABBB123";
```

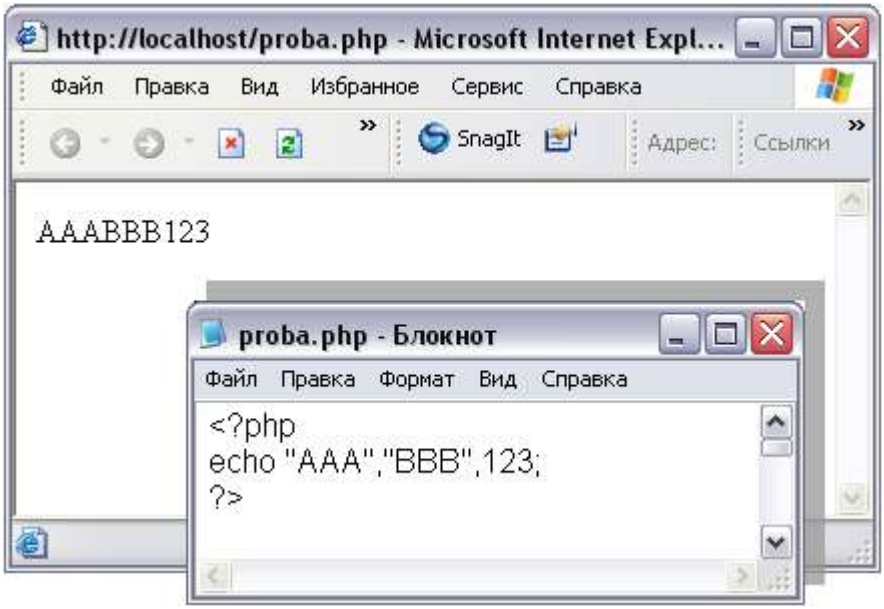

Рис. 2.1. Вывод нескольких элементов данных с помощью оператора echo

Особо следует отметить вывод строковых данных, которые заключаются в двойные или одинарные кавычки. Если вы хотите вывести произвольную последовательность символов с помощью оператора echo, то ее следует заключить в кавычки. Внутри этих кавычек могут быть любые символы (буквы, цифры и т.п.), в том числе и имена переменных (см. разд. 2.3). В случае двойных кавычек строка символов будет интерпретироваться PHP. Если, например, в ней найдутся имена переменных, то в отображаемой строке эти имена переменных будут заменены их значениями. Более того, если в этой строке найдутся теги HTML (дескрипторы, заключенные в уголковые скобки), то браузер отобразит этот HTML-код так, как он и должен это делать при восприятии HTML-документов. Если же вы хотите отобразить строку символов так, как вы ее написали, без какой ни было интерпретации, то ее следует заключить в одинарные кавычки. В листинге 2.1 приведен тестовый PHP-код, результат которого отображается в браузере, как показано на рис. 2.2.

Листинг 2.1. Пример использования оператора echo

```
<?php
$x="Привет"; // присвоение значения переменной
echo $x, " всем!", "<br>";
echo "$x всем!<br>";
echo '$x всем!<br>'
?>
```

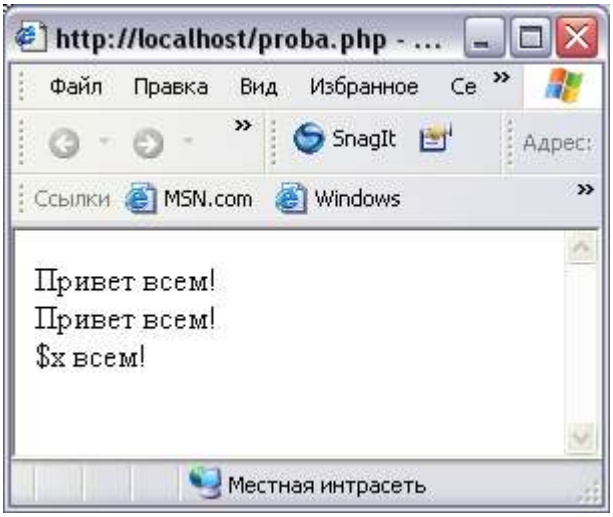

Рис. 2.2. Пример вывода с помощью оператора echo

Здесь использован оператор присваивания значения переменной $x=”Привет”. Подробнее о переменных и присваивании им значений рассказано в разд. 2.3. Далее в сценарии следуют три оператора echo. Первые два из них дают одинаковые результаты, а третий оператор echo выводит строку символов, заключенную в одинарные кавычки. Элемент $x кода PHP в этой строке не интерпретируется (т.е. не заменяется его значением), а выводится просто как последовательность символов.

> ***Внимание***
> Способ отображения символьных строк существенно зависит от того, в двойных или одинарных кавычках заключены символы строки.

2.2. Типы данных

PHP относится к языкам с так называемыми *свободными типами данных*. Это означает, что допустимо применение одних и тех же операций к данным различных типов, а также присвоение одной и той же переменной данных то одного, то другого типа. Этой возможностью следует пользоваться осторожно во избежание недоразумений.

Тип данных — некоторое ограничение на множество возможных значений. Если мы каким либо образом выделяем (ограничиваем) некоторое множество значений, то тем самым мы определяем тип данных.

В табл. 2.1 приведены основные (примитивные) типы данных, которое определены в PHP.

Таблица 2.1. *Типы данных в PHP*

Тип данных		Примеры	Описание значений
Строковый или символьный (String)		"Привет всем!" "д.т. 123-4567" 'Сегодня 24.01.2006г.'	Последовательность символов, заключенная в кавычки, двойные или одинарные
Числовой	Целочисленный (integer)	−567 25 0	Число, последовательность цифр, перед которой может быть указан знак числа (+ или −); перед положительными числами не обязательно ставить знак +. Число записывается без кавычек
	С плавающей точкой (float)	5.47 12.5E2 35E-3	Число с дробной частью. Целая и дробная части чисел разделяются точкой. В экспоненциальной форме символ E используется для обозначения 10, за которым следует число, указывающее степень. Например, запись 3E2 означает $3 \cdot 10^2$, т.е 300
Логический (булевский, Boolean)		true false	Этот тип имеет два значения: true (истина, да) или false (ложь, нет)
Null		null	Этот тип данных имеет одно значение — null, указывающее на отсутствие какого бы то ни было значения
Массив (Array)			Этот тип данных имеет одно множество значений, которые могут быть различных типов.
Объект (Object)			Программный объект, определяемый своими свойствами.
Ресурс (Resourse)			

Внимание

Ключевые слова true, false и null могут использоваться в выражениях PHP в любом регистре.

В выражениях с операторами могут использоваться данные различных типов. Например, в арифметическом выражении могут оказаться данные не только

числового, но и строкового или логического типа. В таких случаях интерпретатор PHP автоматически приводит данные к нужному типу.

Например, в выражении с арифметическим оператором сложения + операнды могут быть как строковыми, так и числовыми:

```
5 + 4                   // результат: 9
“5” + ”4”               // результат: 9
“5” + 4                 // результат: 9
“5рублей” + 4           // результат: 9
“Доход 5” + 4           // результат: 4
“Привет” + 4            // результат: 4
“Привет” + ”всем”       // результат: 0
5 + true                // результат: 6
5 + false               // результат: 5
“Привет” + true         // результат: 1
“Привет” + false        // результат: 0
```

Если строка не содержит в качестве своих первых символов цифры (точнее, числ перед которым может быть указан знак), то в арифметических выражениях она преобразуется в число 0.

Данные различных типов могут участвовать в логических операциях и операциях сравнения, которые возвращают логическое значение true или fals. В таких случаях интерпретатор PHP автоматически приводит данные нелогического типа к логичекому. При этом следующие значения преобразуются в false:

- ❒ Строка “FALSE”, не зависимо от регистра
- ❒ Пустая строка “” и строка “0”
- ❒ Число 0, целое или с плавающей точкой
- ❒ Null
- ❒ Пустой массив (количество элементов равно 0)
- ❒ Пустой объект (количество элементов равно 0)

Другие значения преобразуются в true.

Значения логического типа могут участвовать в арифметических операциях. В таких случаях значение false преобразуется в 0, а true — в 1. Например, выражение true+true возвращает 2, а true+false — 1.

Иногда требуется данные некоторого типа привести к определенному типу, чтобы гарантированно обеспечить корректность последующих операций над ними, не полагаясь на возможности автоматического преобразования типов. Для приведения

данных к заданному типу служат специальные операторы — имя типа в круглых скобках, записываемое перед данными, которые необходимо преобразовать:

- (int) или (integer) — приведение к целочисленному типу;
- (float), (double) или (real) — приведение к числовому типу с плавающей точкой;
- (string) — приведение к строковову типу;
- (bool) или (boolean) — приведение к логическому (булевскому) типу;
- (array) — приведение к типу массив;
- (object) — приведение к типу объект.

Примеры:

(int) 12.65 // результат: 12

(float) “12.65 руб.” // результат: 12.65

(bool) “12.65 руб.” // результат: true

Перечисленные выше операторы преобразования типов применимы и к переменным, и к вызовам функций, о чем будет рассказано в следующих разделах.

2.3. Переменные и оператор присваивания

2.3.1. Имена переменных

Переменная является контейнером для хранения данных. Данные, сохраняемые в переменной, называются *значениями* этой переменной. Переменная имеет имя (идентификатор) — последовательность букв, цифр, символов подчеркивания без пробелов и знаков препинания, начинающаяся обязательно с буквы или символа подчеркивания. Перед именем переменной обязательно пишется символ доллара ($).

По существу префикс $ в имени переменной указывает интерпретатору, что он имеет дело с переменной, а не чем-либо иным. Поэтому в имени переменной можно использовать и ключевые слова языка, такие как if, else, while и т.д.

Примеры правильных имен переменных: $`myName`, $`my_adress`, $`_x`, $`tel123_4567`, $`var`, $Сумма, $if.

Примеры неправильных имен переменных: $`512group`, $`my adress`, `tel:412 3456`, $`Сумма$`.

> ***Внимание!***
> PHP является регистрозависимым языком относительно имен переменных и констант. Это означает, что изменение регистра символов (с прописных на строчные и наоборот) в имени переменной приводит к другой переменной. Например, $`variable`, $`Variable` и $`vaRiabLe` — различные переменные. Однако ключевые слова языка

(if, while, function и т.д.), а также имена функций могут использоваться в любом регистре. В отличие от PHP, язык JavaScript является полностью регистрозависимым.

2.3.2. Создание переменных

Переменная создается при первом ее упоминании в программном коде с использованием оператора присваивания ей некоторого значения. Оператор присваивания обозначается знаком равенства (=). Слева от него указывается имя переменной, а справа — выражение языка, возвращающее в результате вычисления некоторое значение. В простейшем случае справа от символа оператора присваивания указывается прстоое значение — число, строка символов в кавычках, ключевое слово, обозначающее специальное значение некоторых типов (true, false, null).

Создать переменную можно с помощью оператора присваивания (=):

```
$var = ”Привет всем”;     /* переменной $var присвоено строковое значение
                           ”Привет  всем” */
$x = 25.2E2;              // переменной $x присвоено числовое значение 2520
$x = true;                // переменной $x присвоено логическое значение true
```

Одной и той же переменной можно присвоить то одно, то другое значение. Тип переменной определяется типом значения, которое она в данный момент имеет (содержит). Так, в рассмотренном выше примере переменная $x сначала имела значение числового типа, а затем — логического.

Не следует путать оператор присваивания с отношением равенства и соответствующей операцией сравнения. Выражение с оператором = интерпретатор вычисляет следующим образом: переменной слева от него присваивается значение, расположенное справа от него.

Значение одной переменной можно присвоить другой переменной:

```
$x = 5;
$y = $x;          // переменная $y имеет значение 5
```

Переменные могут участвовать в выражениях с операторами:

```
$x = 5;
$y = $x + 2;      // переменная $y имеет значение 7
```

Когда мы пишем $y=$x, то тем самым выражаем следующее: значение переменной $x присвоить переменной $y. Это так называемое присвоение по значению, при котором значение одной переменной копируется в другую переменную. Однако кроме этого способа в PHP возможно организовать присвоение по ссылке, при котором копирование значения не происходит, а новая переменная становится псевдонимом (еще одним именем) для той переменной, на которую она ссылается.

Для присвоения значения по ссылке используется символ & (амперсанд). Так, в выражении $x=&$y; переменная $x ссылается на переменную $y. При этом обращение к переменной $x возвращает значение переменной $y, а изменение значения $x отражается на значении $y, как и, наоборот, изменение значения $y вызывает изменение значения $x:

```
<?php
$x = "Саша";
$y = "Маша";
$x = &$y;          // значение $x равно “Маша”
$x = "Вася";       // значения $x и &y равны “Вася”
/* Вывод значений */
echo '$x=', $x;    // вывод строки: $x=Вася
echo "<br>";       // перевод строки
echo '$y=', $y;    // вывод строки: $y=Вася
?>
```

> ***Примечание***
> Присвоение значений переменным по ссылке происходит быстрее, чем по значению и это становится заметным, например, при установке значений элементам больших массивов с помощью оператора цикла.

У существующей переменной можно удалить текущее значение, т.е. сделать ее без какого бы то ни было значения (очистка переменной и занимаемой ею памяти). При этом переменная продолжает существовать. С этой целью используется функция unset(*имя_переменной*):

```
$x = 5;
unset($x);          // удаление значения переменной $x
```

В действительности в результате выполнения функции unset($x) переменная $x приобретает значение null.

> ***Примечание***
> В данной главе мы еще не рассматривали функции как таковые. Пока вы можете считать, что функция (точнее ее вызов в программе) задается именем, за которым следует пара круглых скобок. В скобках, если это требуется для данной функции, указываются параметры — значения, передаваемые функции из внешнего программного кода. Функция обычно (но не обязательно) что-то возвращает и в этом случае ее можно использовать в выражениях языка.

Как уже отмечалось, тип переменной определяется типом значения, которое она имеет в текущий момент времени. Однако в PHP имеются операторы приведения типов (см. разд. 2.2). Обычно они используются, чтобы обеспечить надлежащее преобразование данных, в результате которого можно рассчитывать на корректное применение тех или иных операций. Напомню, что и без этих операторов

интерпретатор PHP, анализируя выражение, преобразует данные к некоторому подобающему в данной ситуации типу. Однако его “интеллектуальные потуги” иногда могут не соответствовать ожиданиям автора программы, и тогда автор может форсировать приведение типов по своему усмотрению. Вот пример:

```
$ x= ”5.3 рублей”;        // строка символов ”5.3 рублей”
$y = $x + 7;              // число с плавающей точкой 12.3
$y = (int)$x + 7;         // целое число 12
$y = (float)$x + 7;       // число с плавающей точкой 12.3
$z = (bool)$y;            // логическое значение true
```

2.3.4. Отображение значений переменных

Значения переменных можно вывести на экран с помощью оператора echo. Этот оператор направляет вывод значений в выходной документ, отображаемый браузером. При выполнении примеров и других экспериментальных кодов на языке PHP, а также при отладке ваших реальных сценариев очень полезно организовать вывод промежуточных и окончательных результатов на экран. Для этого мы используем оператор echo (см. разд. 2.1). Вывод текущих значений переменных можно сделать несколькими способами. Приведу лишь наиболее употребительные:

- echo $var1, $var2, …, $varN; — выводятся значения переменных, перечисленные в операторе echo через запятую; при отображении значения переменных следуют друг за другом без пробелов (запятые между именами переменных не отображаются).
- echo “Переменные: $var1, $var2, …, $varN”; — выводится символьная строка (заключенная в двойные кавычки), в которой имена переменных заменяются их значениями.
- echo ‘Переменные: $var1, $var2, …, $varN’; — выводится строка так, как есть, т.е. выодится все, что указано внутри одинарных кавычек без подстановок значений переменных.

Замечу попутно, что в строку символов, подлежащую выводу в окно браузера, можно вставлять теги HTML, которые позволяют отформатировать выводимый текст. В листинге 2.2. представлен PHP-код, который иллюстрирует приведенные выше замечания, а результат его выполнения, отображенный в браузере Internet Explore, показан на рис. 2.3.

Листинг. 2.2. Пример вывода переменных в окно браузера

```
<?php
$var1="Привет";
$var2=" всем!";
echo  $var1, $var2; // вывод значений нескольких  переменных
```

```
echo "<br>Переменные: $var1, $var2<br>";
echo 'Переменные: $var1, $var2';
?>
```

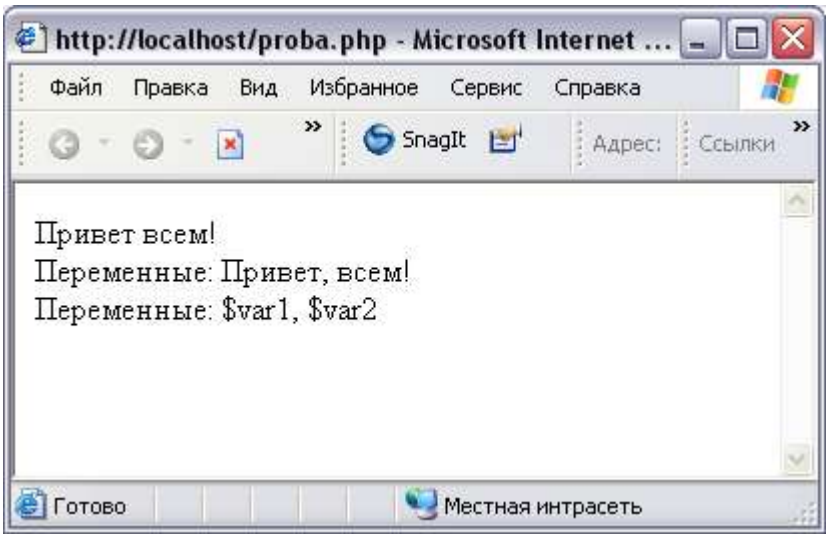

Рис. 2.3. Пример отображения значений переменных

> ***Внимание***
> Вывод с помощью оператора echo строки, в которой указаны переменные, существенно зависит от того, в какие кавычки заключена эта строка символов — двойные или одинарные. В случае двойных кавычек переменные вычисляются, а вместо их имен в строку подставляются значения. В случае одинарных кавычек содержимое строки не анализируется и не вычисляется интерпретатором PHP, а выводится как есть.

Значения некоторых типов отображаются оператором echo не всегда так, как они были заданы в коде программы. Так, логическое значение true отбражается как 1, а false — не отображается вовсе; числовые значения отображаются в экспонециальной или неэкспоненциальной форме в зависимости от их величины (рис. 2.4). Вместе с тем вы можете отформатировать выводимые сообщения по своему усмотрению, но об будет рассказано в разд. 2.6.

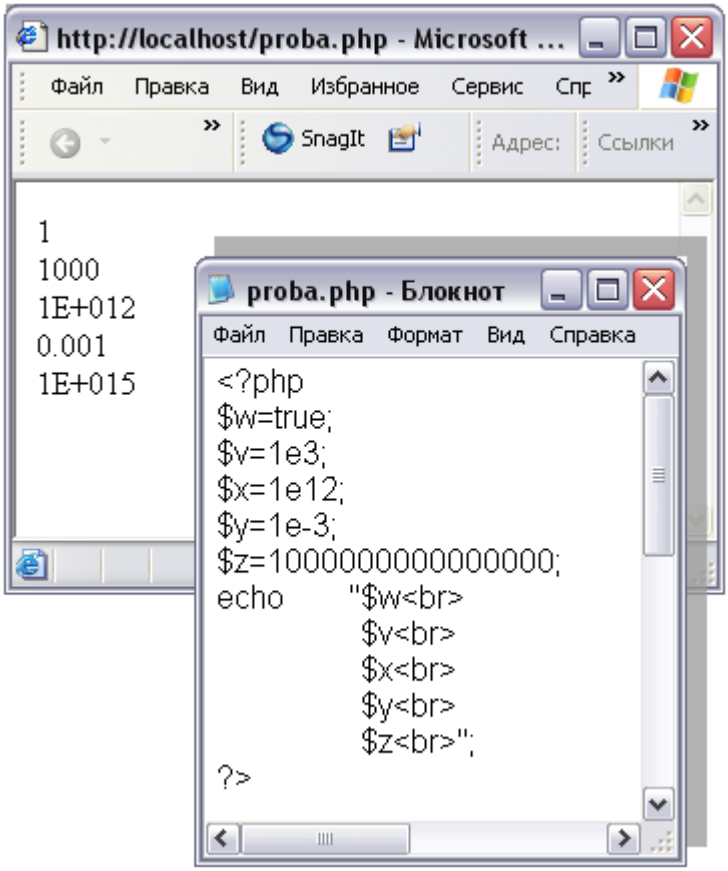

Рис. 2.4. Отображение логического и числовых значений с помощью оператора echo

Вывод значений элементов массива в составе текстовых строк имеет некоторую особенность: элементы массива необходимо заключать в фигурные скобки (подробнее об этом см. разд. 2.9.3).

Для вывода более подробной информации о переменных, которая может понадобиться при отладке программы, служит функция var_dump(*список_переменных*). В списке переменных указывается одно или несколько имен переменных, разделенных запятыми. Эта функция ничего не возвращает. На рис. 2.5 показан пример использования var_dump() для переменных различных типов. Так, в случае строковой переменной выводится ее тип длина значения и само значение, для числовых и логических переменных выводится тип и значение. Данная функция особенно полезна при выводе информации о массивах. В этом случае выводится тип переменной (array), количество элементов массива и список сведений об элементах вида `индекс => элемент`. Индекс может быть как числовым, так и символьным в зависимости от того, как он был задан в определении массива. Подробнее о массивах см. разд. 2.9.

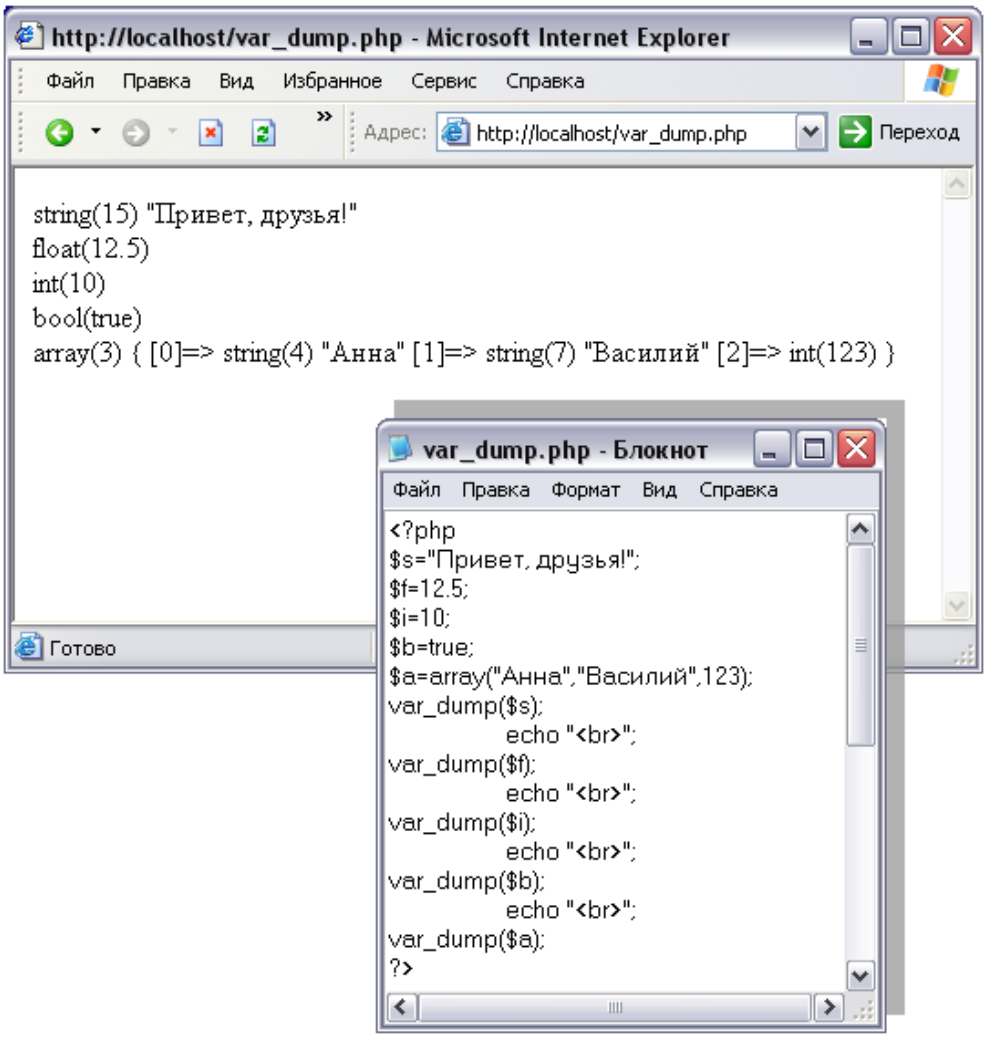

Рис. 2.5. Вывод информации о переменных с помощью функции var_dump()

Менее информативной, чем var_dump(), функцией вывода информации о переменных является print_r(*список_переменных*). Для переменных типа массив эта функция выводит список вида индекс => элемент без указания типов и других характеристик элементов. Для переменных других типов выводятся только их значения. Пример использования этой функции можно найти в разд. 2.7.3.

2.3.3. Переменные переменные

В программах иногда может понадобиться интерпретировать строковое значение как имя переменной. Иначе говоря, мы хотим присвоить некоторой переменной в

качестве значения имя другой переменной. Так можно создавать переменные динамически, т.е. програмно задавать переменные, которые необходимы в данный момент вычислительного процесса. Другими словами, переменная может содержать другую переменную, называемую “переменной переменной”. Для создания переменной переменной используется двойной символ $ в качестве префикса ее имени. Так, запись $$x означает переменную, имя которой указано в качестве значения переменной $x.

На рис. 2.6. показаны код программы в окне текстового редактора и результат его выполнения в окне браузера. Здесь переменная $x содержит строковое значение “Девочка”. Выражение $$x=”Маша” означает, что переменной, имя которой содержится в переменной $x, присваивается значение “Маша”. Таким образом, создается переменная $Девочка и ей присваивается значение “Маша”. С помощью оператора echo выводится строка, содежащая значение переменной $x и динамически созданной переменной $Девочка.

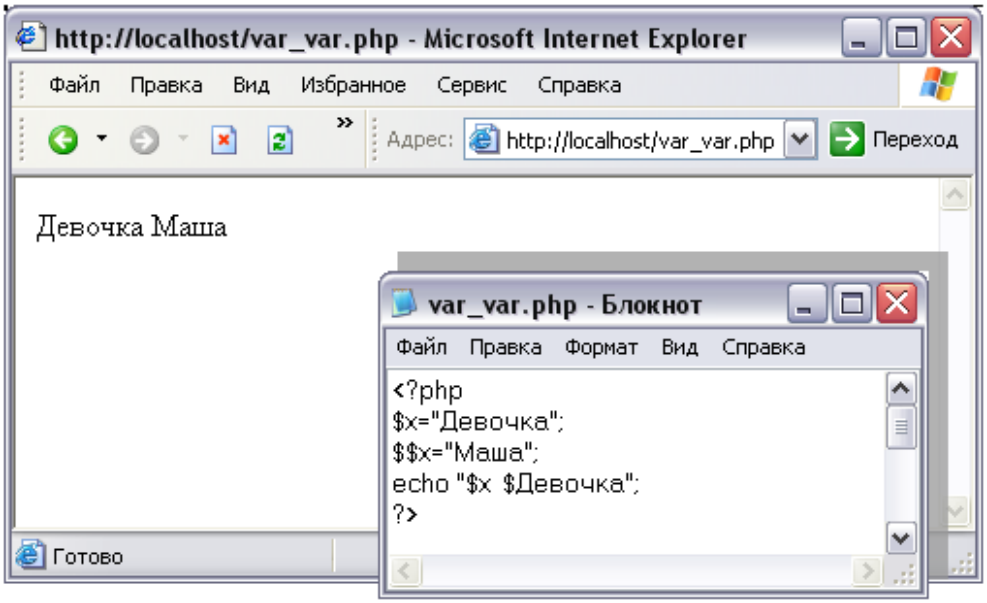

Рис. 2.6. Пример использования переменных переменных

В рассматриваемом примере оператор echo “$$x” выведет строку “$Девочка”, оператор echo “${$x}” — строку “Маша”. Здесь использовались фигурные скобки, чтобы указать интерпретатору, что требуется вывести, а именно значение переменной $Девочка: значение переменной {значение переменной x}.

На рис. 2.7 показан еще один пример, в котором над переменными переменными производятся арифметические действия.

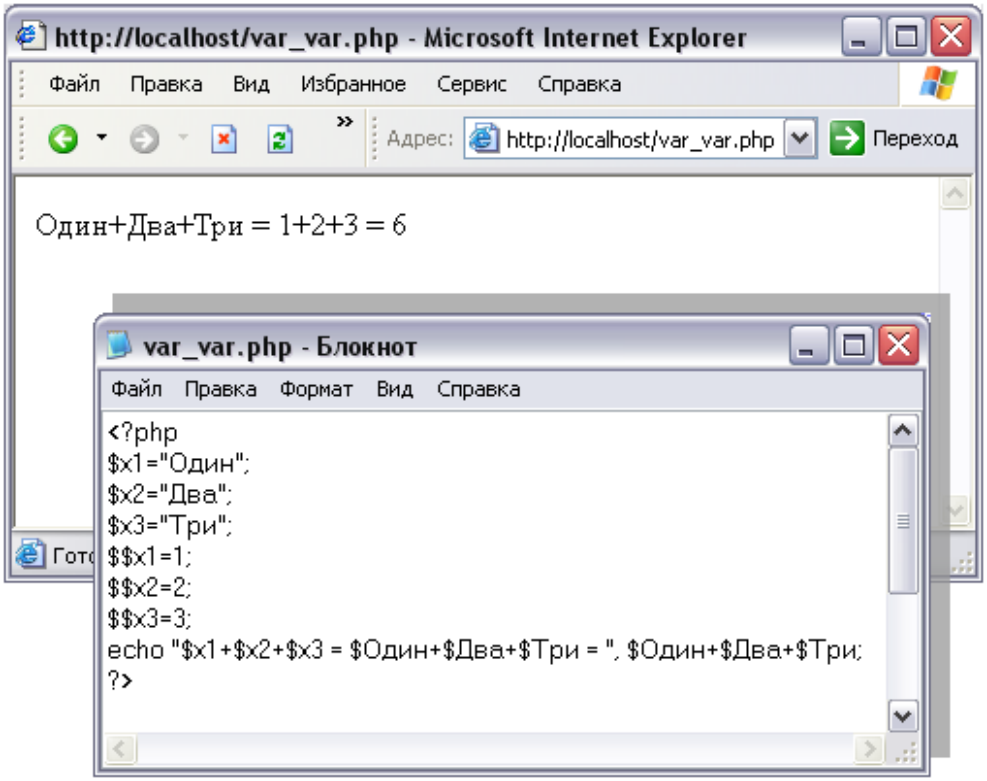

Рис. 2.7. Пример вычислений над переменными переменными

2.3.5. Область действия переменных

Областью действия (видимости, доступности) переменной является тот программный код из одного и того же файла, в котором она была определена. При этом она также видна и в PHP-кодах, включаемых в данный из внешнего файла с помощью функции include(). Однако внутри кода функций такие переменные не видны, если не предпринять специальных мер.

В следующем примере переменная $x видна во всем программном коде, включая и код из файла mylib. php. Если код из этого файла не изменяет значение переменной $x, то выведенное значение будет равно 9.

```
<?php
$x=5;
?>
Далее выводим переменную $x:
<?php
```

```
include("mylib.php"); // включение файла
$x=$x+4;
echo "$x"; // вывод значения переменной $x
?>
```

Таким образом, переменные в PHP были бы глобальными в полном смысле этого слова, если бы не одно обстоятельство. Дело в том, что программный код может содержать определения и вызовы функций. Если созданная во внешнем коде переменная используется еще и в коде функции, то там она будет не глобальной, как ожидается, а локальной. Это означает, что значение переменной, заданное во внешнем коде, не видно внутри кода функции. С другой стороны, изменения значений этой переменной внутри функции не отразятся на этой же переменной во внешнем коде. Чтобы сделать переменные, созданные во внешнем коде, доступными и внутри кода функций (т.е. обеспечить им настоящую глобальность), они должны быть объявлены глобальными в коде функций с помощью ключевого слова global. Кроме того, переменные внутри функций могут быть объявлены как статические (static). Статическая переменная является локальной, но сохраняет свое значение и при следующем вызове функции, где она была определена, данное значение может быть использовано. Примером такой функции является strtok()(см. разд.2.6.3).

Примечание1

В языках JavaScript и C глобальные переменные автоматически доступны и внутри функций, если только они не были определены там как локальные. Т.е. глобальные переменные, видны везде. Так, в JavaScript локализация переменной в функции задается посредством ключевого слова var. Если, например, переменная myvar является глобальной, то чтобы в коде функции использовать это же имя для локальной переменной, достаточно там написать выражение var myvar.

Примечание2

В PHP имеется множество предопределенных переменных типа массив. Эти переменные глобальны в полном смысле этого слова и содержат сведения о Web-сервере, окружении и данных, которое сервер получил от клиента (браузера).

Примечание 3

Базовый оператор присвоения знаяений обозначется знаком равенства. Однако допускаются и специальные операторы присвоения значений, которые в ряде случаев оказываются более удобными. О них рассказывается в разд. 2.5.4

2.3.6. Проверка существования переменных и их типов

Иногда требуется проверить, определена ли та или иная переменная. Для этого служит функция

isset(*имя_переменной1, имя_переменной2, ..., имя_переменнойN*)

Если все переменные в списке существуют, то функция возвращает true, в противном случае — false.

Чтобы выяснить, является ли значение переменной пустым, используется функция

empty(*имя_переменной*)

Эта функция возвращает true, если она не объявлена, является пустым массивом или значение переменной равно:

- 0 — целое число
- “” — пустая строка
- “0” — строка с 0
- false
- null, переменная не объявлена или является пустым массивом.

В противном случае функция empty() возвращается false.

> ***Примечание***
> Начиная с версии 5 PHP, объект без свойств не считается пустым с точки зрения функции empty().

Чтобы проверить тип переменной, используются следующие функции:

- is_string(*имя_переменной*) — возвращает true, если переменная является строковой и false — в противном случае;
- is_int(*имя_переменной*) — возвращает true, если переменная является целочисленной и false — в противном случае;
- is_float(*имя_переменной*) — возвращает true, если переменная является числовой с плавающей точкой и false — в противном случае;
- is_numeric(*имя_переменной*) — возвращает true, если переменная является числовой (типа integer или float), а также строкой, содержащей только числа. В противном случае возвращается false;
- is_null(*имя_переменной*) — возвращает true, если переменная имеет значение null и false — в противном случае;
- is_array(*имя_переменной*) — возвращает true, если переменная является массивом (имеет тип array) и false — в противном случае.

2.4. Константы

Константы похожи на переменные, поскольку имеют имена (идентификаторы) и значения. Значения присваиваются константам однажды в коде программы и после уже не могут быть изменены. Константы имеют следующие особенности:

- Имя константы образуется так же, как и имя переменной, за исключением того, что не указывается префикс $. Этот символ указывает, что за ним следует имя переменной, а мы сейчас имеем дело с константой. Поскольку имени константы не должен предшествовать символ $, в качестве имени константы нельзя использовать ключевые слова PHP, такие как if, else, while, echo, default, function и т.п.
- Значения констант могут принадлежать только типам string, integer, float, и boolean.

- Создание констант (объявление имени и присвоение значений) обеспечивается специальной функцией `define("имя_константы", значение)`. Например, `define("MYNAME","Дунаев Вадим Вячеславович")`. Обычный для переменных оператор = присваивания значений здесь не годится.

Константы полезны, если вы хотитете обезопасить себя от случайного изменения в программе данных, которые должны быть постоянными. Если же вам все же потребуется их изменить, то это придется делать в одном месте, а именно там, где вы их однажды определили с помощью функции define(). Кроме того, вы можете создавать один и тот же программный код, который должен работать с различными наборами исходных данных, адаптированных, например, к конкретному пользователю, среде выполнения, варианту реализации и т.п. В этом случае использование констант поможет улучшить технологичность вашего проекта. На рис. 2.8. показан пример определения и вывода констант.

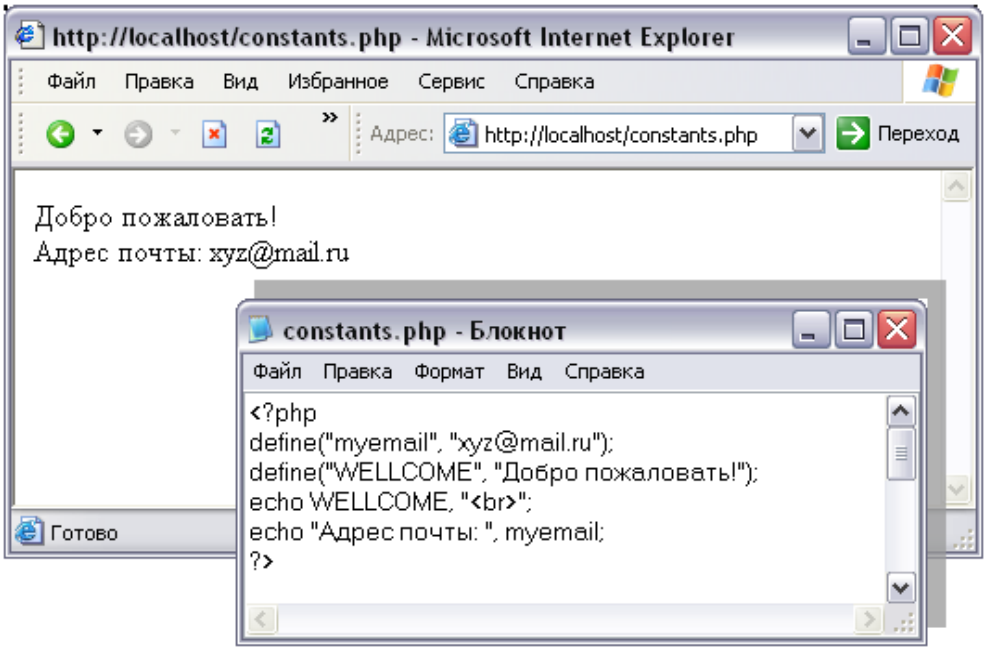

Рис. 2.8. Пример определения и вывода констант

При использовании констант в строке, заключенной в двойные кавычки, подстановка их значений вместо имени (как в случае переменных) не происходит.

Примечание

В PHP имеется множество предопределенных констант. Например, константа __FILE__ (окаймляется двойными символами подчеркивания) содержит полное имя текущего PHP-файла, а константа __LINE__ содержит номер текущей строки в PHP-файле. Математические константы приведены в разд. 2.8.2.

Иногда требуется проверить, определена ли та или иная константа. Для этого существует функция

defined(*имя_константы1, имя_ константы2, ..., имя_ константыN*)

Если все константы в списке существуют, то функция возвращает true, в противном случае — false.

Примечание
Аналогичная функция для проверки существования переменных — isset().

2.5. Операторы

Операторы предназначены для составления выражений. Оператор применяется к одному или двум данным, которые в этом случае называются *операндами*. Например, оператор сложения применяется к двум операндам, а оператор логического отрицания — к одному операнду. *Элементарное выражение*, состоящее из операндов и оператора, выполняется интерпретатором и может иметь некоторое значение. В этом случае говорят, что оператор возвращает значение. Оператор имеет тип, совпадающий с типом возвращаемого им значения. Элементарное выражение с оператором и операндами, возвращающее значение, можно присвоить переменной.

В предыдущем разделе мы рассмотрели оператор присваивания. Сейчас мы обратимся к другим операторам, предназначенным для работы с данными различных типов.

2.5.1. Комментарии

Операторы комментария позволяют выделить фрагмент кода, который не выполняется интерпретатором, а служит лишь для пояснений содержания программы. Не пренебрегайте комментариями при написании своих программ.

В PHP допустимы три вида оператора комментария:

- `//` — одна строка символов, расположенная справа от этого оператора, считается комментарием;
- # — действует так же, как и оператор //;
- `/*...*/` — все, что заключено между `/*` и `*/`, считается комментарием; с помощью этого оператора можно выделить несколько строк в качестве комментария.

2.5.2. Арифметические операторы

Арифметические операторы, такие как сложение, умножение и т. д., в PHP предназначены для манипулирования числовыми данными, хотя могут применяться к данным любых типов. В таблице 2.2. приведены арифметические операторы.

Таблица 2.2. *Арифметические операторы*

Оператор	Название	Пример
+	Сложение	$X+$Y
-	Вычитание	$X-$Y
*	Умножение	$X*$Y
/	Деление	$X/$Y
%	Деление по модулю (остаток от деления)	$X%$Y
++	Увеличение на 1	$X++ ++$X
--	Уменьшение на 1	$Y-- --$X

Первые четыре оператора не требуют особых разъяснений. Оператор деления (/) всегда возвращает число типа float, даже если в результате получается целое.

Оператор деления по модулю возвращает остаток от деления первого числа на второе, если первое число не меньше второго. В противном случае возвращается первое число. Например, 10%4 возвращает 2, 3%5 — 3, 12%12 — 0. При этом результат будет отрицательным числом, если первый из операндов отрицателен. Например, -5%2 возвращает -3, 5%-2 — 3.

Операторы ++ и -- сочетают в себе действия операторов соответственно сложения и вычитания, а также присваивания. Выражение $X++ эквивалентно выражению $X=$X+1, однако оно возвращает не новое, а прежнее значение переменной $X. Таким образом, оператор $X++ сначала возвращает текущее значение $X, а затем увеличивает его на 1. В противоположность этому оператор ++$X сначала увеличивает значение переменной $X на 1, а затем возвращает новое значение. Аналогично, оператор $X-- сначала возвращает текущее значение $X, а затем уменьшает его 1. Оператор --$X l делает это в противоположном порядке.

Операторы ++ и -- называют соответственно *инкрементом* и *декрементом*.

Символ - используется не только как бинарный (для двух операндов) оператор вычитания, но и как унарный (для одного операнда) оператор отрицания для указания того, что число является отрицательным.

Однако формально ничто не мешает нам применить эти операторы к данным нечисловых типов. При этом все нечисловые операнды автоматически преобразуются в числа. Например, в случае строковых данных интерпретатор пытается привести их к числовому типу. Если префиксом строки является число, то эта строка будет преобразована в соответствующее число, в противном случае она

будет преобразована в число 0. Логические значения преобразуются в 1 и 0 соответственно для true и false. Подробнее об этом см. разд. 2.3.

Примеры:

```
5+3                  // результат: 8
"5"+"3"              // результат: 8
"5"+3                // результат: 8
"5рублей"+3          // результат: 8
"Доход 5"+3          // результат: 3
"Привет"+3           // результат: 3
"Привет"+"всем"      // результат: 0
$x=8;
$y=3;
$x/$y;               // результат: 2.66666666667
$x%$y;               // результат: 2
$x++;                /* результат: значение переменной $x становится
                     равным 9, но выражение возвращает 8 */
++$x;                /* результат: значение переменной $x становится
                     равным 9, и выражение возвращает 9 */
```

Примечание

В JavaScript в выражениях с оператором сложения + данных строкового и числового типа последние приводятся к строковому, а не числовому типу. Поэтому, выражение “Вася”+5 даст в результате “Вася5”, а не 5, как в PHP. С другой строны, в PHP выражение “5”+4 вернет 9, а в JavaScript — “54”.

2.5.3. Строковый оператор

Две строки можно объединить (склеить) с помощью операции конкатенации, обозначаемой точкой. Применение этой операции к двум строкам дает в результате строку, полученную путем приписывания справа второй строки к первой. Например, выражение `"Саша"."Маша"` возвращает в результате `"СашаМаша"`.

Пример:

$x=”Привет”;

$y=”всем”;

echo $x.” ”.$y;

Здесь склеиваются три строки: значение переменной $x, строка с пробелом и значение переменной $y.

Примечание

В JavaScript оператор конкатенации обозначается так же, как и оператор сложения чисел (+).

2.5.4. Дополнительные операторы присваивания

Кроме обычного оператора присваивания = имеются еще шесть дополнительных операторов, сочетающих в себе действия обычного оператора присваивания и арифметических операторов (табл. 2.3).

Таблица 2.3. *Операторы присваивания*

Оператор	Пример	Эквивалентное выражение
+=	X+=Y	X=X+Y
-=	X-=Y	X=X-Y
*=	X*=Y	X=X*Y
/=	X/=Y	X=X/Y
%=	X%=Y	X=X%Y
.=	X.=Y	X=X+Y

Пример:

```
$x="Привет"
$x.=" всем"   // "Привет всем"
$x=5; $x*=7     // 35
```

2.5.4. Операторы сравнения

Результатом вычисления элементарного выражения, содержащего оператор сравнения и операнды (сравниваемые данные), является логическое значение, т. е. `true` или `false`. Операторы сравнения приведены в таблице 2.4

Таблица 2.4. *Операторы сравнения*

Оператор	Название	Пример
==	Равно	$X==$Y
===	Тождественно равно	$X===$Y
!=	Не равно	$X!=$Y
<>		$X<>$Y
!===	Не равно тождественно	$X!===$Y
>	Больше, чем	$X>$Y
>=	Больше или равно (не меньше)	$X>=$Y
<	Меньше, чем	$X<$Y
<=	Меньше или равно (не больше)	$X<=$Y

Внимание

Оператор "равно" записывается с помощью двух символов = без пробелов между ними, а оператор "тождественно равно" — с помощью трех подряд идущих символов =.

Сравнивать можно данные различных типов. Что касается до оператора равенства (неравенства), то допустимы две их разновидности:

- Гибкий оператор равенства (==). Выражение возвращает true, если операнды равны. В противном случае возвращается false. При этом типы сравниваемых данных могут разичаться.
- Жесткий оператор равенства (===). Выражение возвращает true, если операнды не только равны, но имеют еще и динаковые типы. В противном случае возвращается false.

Примеры:

```
"5.3" == 5.3          // возвращает true
"5.3" === 5.3         // возвращает false
"1" == true           // возвращает true
1 == true             // возвращает true
"1" == "true"         // возвращает false
1 == "true"           // возвращает false
"0" == "false"        // возвращает false
"0" == false          // возвращает true
0 == "false"          // возвращает true
0 == false            // возвращает true
0 === "false"         // возвращает false
null==null            // возвращает true
null===null           // возвращает true
"abcd" > " abcd"      // возвращает true
"abc" < "abcd"        // возвращает true
"235ab" < "abcdxyz"   // возвращает true
"235xyz" < "abc"      // возвращает true
```

Внимание

При практическом программировании следует обратить особое внимание на сравнение логических значений true и false с данными других типов, таких как пустые строки, null, “0”, 0 и т.д. Многие функции возвращают false тогда, когда что-то не так (например, возникла ошибка преобразования данных, произошел выход за границы массива). Подобные ситуации обычно каким-либо образом обрабатываются в программе и при этом чаще всего проверяется условие, а не равен ли результат false. Здесь важно корректно сформулировать проверяемое условие.

2.5.6. Логические операторы

Логические данные, обычно получаемые с помощью элементарных выражений, содержащих операторы сравнения, можно объединять в более сложные выражения. Для этого используются логические (булевские) операторы — логические союзы И,

ИЛИ, исключающее ИЛИ а также оператор отрицания НЕ (табл. 2.5). Выражения с логическими операторами возвращают значение `true` или `false`.

Таблица 2.5. *Логические операторы*

Оператор	**Название**	**Пример**
`!`	Отрицание (НЕ)	`!$X`
`&&` `and`	И	`$X&&$Y`
`||` `or`	ИЛИ	`$X\|\|$Y` `$X or $Y`
`xor`	Исключающее ИЛИ	`$X xor $Y`

Оператор отрицания `!` применяется к одному операнду, изменяя его значение на противоположное: если $`X` имеет значение `true`, то `!$X` возвращает значение `false`, и наоборот, если `X` имеет значение `false`, то `!X` возвращает значение `true`. Логическим союзам И и ИЛИ соответствуют по два оператора. Они действуют одинаково, но имеют различные приоритеты, определяющие порядок их вычисления в сложных выражениях. Приоритеты логических операторов определяются их положением в последовательности: `!, &&, ||, and, xor, or`.

В табл. 2.6 указано, какие значения возвращают операторы И, ИЛИ и xor при различных логических значениях двух операндов.

Таблица 2.6. *Примеры использования операторов И, ИЛИ*

$X	**$Y**	**$X && $Y**	**$X || $Y**	**$X xor $Y**
`true`	`true`	`true`	`True`	`false`
`true`	`false`	`false`	`true`	`true`
`false`	`true`	`false`	`true`	`true`
`false`	`false`	`false`	`false`	`false`

Операторы `&&` и `||` еще называют *логическим умножением* и *логическим сложением* соответственно, или конъюнкцией и дизъюнкцией соответственно.

Примеры:

```
$x=false || 2*2==4;         // значение переменной x равно true
$x=5<2 || "abcd" <="xy";    // значение переменной x равно true
$y=!$x;                     // значение переменной y равно false
$z=$x && $y;                // значение переменной z равно false
$z=$x || $y;                // значение переменной z равно true
```

```
$x=true; $y=false;
$z=$x xor !$y;            // значение переменной z равно false
```

Сложные логические выражения, состоящие из нескольких более простых, соединенных операторами И, ИЛИ и исключающее ИЛИ, выполняются по так называемому *принципу короткой обработки*. Дело в том, что значение всего выражения бывает можно определить, вычислив лишь одно или несколько более простых выражений, не вычисляя остальные. Данное обстоятельство требует особого внимания при тестировании сложных логических выражений. Так, если какая-нибудь составная часть выражения содержит ошибку, то она может остаться невыявленной, поскольку эта часть выражения просто не выполнялась при тестировании.

2.5.7. Побитовые операторы

Побитовые (поразрядные) операторы применяются к целочисленным значениям и возвращают целочисленные значения. При их выполнении операнды предварительно приводятся к двоичной форме представления, в которой число является последовательностью из нулей и единиц длиной 32. Эти нули и единицы называются *двоичными разрядами* или *битами*. Далее производится некоторое действие над битами, в результате которого получается новая последовательность битов. В конце концов, эта последовательность битов преобразуется к обычному целому числу — результату побитового оператора. В табл. 2.7 приведен список побитовых операторов.

Таблица 2.7. *Побитовые операторы*

Оператор	Название	Левый операнд	Правый операнд
`&`	Побитовое И	Целое число	Целое число
`|`	Побитовое ИЛИ	Целое число	Целое число
`^`	Побитовое исключающее ИЛИ	Целое число	Целое число
`~`	Побитовое НЕ	—	Целое число
`<<`	Смещение влево	Целое число	Количество битов, на которое производится смещение
`>>`	Смещение вправо	Целое число	Количество битов, на которое производится смещение

Операторы `&`, `|`, `^` и `~` напоминают логические операторы, но их область действия — биты, а не логические значения. Оператор `~` изменяет значение бита на противоположное: 0 на 1, а 1 — на 0. Действие побитовых операторов такое же, как и в языке JavaScript (см. разд. 2.15.1).

2.5.8. Операторы условного перехода

Управление вычислительным процессом производится с помощью операторов условного перехода if, switch, и ?: Они обеспечивают выполнение того или иного фрагмента кода в зависимости от выполнения некоторого условия.

Оператор *if*

Оператор условного перехода `if` позволяет реализовать структуру условного выражения:

если ..., то ..., иначе ...

Синтаксис оператора `if` перехода следующий:

```
if (условие)
  {код, который выполняется, если условие истинно}
else
  {код, который выполняется, если условие ложно}
```

В фигурных скобках располагается блок кода — несколько выражений, разделенных точкой с запятой, как это обязательно в PHP. Если в блоке используется не более одного выражения, то фигурные скобки можно не писать. Часть этой конструкции, определяемая ключевым словом `else` (иначе), необязательна. В этом случае остается только часть, определенная ключевым словом `if` (если):

```
if (условие){
  код, который выполняется, если условие истинно
}
```

Вместо ключевого слова else можно использовать elseif (*условие*), чтобы устоновить проверку еще одного условия:

```
if (условие1){
       код, который выполняется, если условие1 истинно
} elseif (условие2){
       код, который выполняется, если условие1 ложно, а условие2 истинно
} else {
       код, который выполняется, если условие1 и условие2 ложны
};
```

Конструкций elseif в операторе if может быть несколько или ни одной.

Оператор *switch*

Оператор `switch` (переключатель) удобен, когда требуется значение переменной или выражения сравнить с определенными величинами и выполнить тот или иной фрагмент кода в зависимости от результата сравнения.

Синтаксис оператора `switch` выглядит следующим образом:

```
switch (выражение) {
```

```
  case вариант1:
    код
    [break]
  case вариант2:
    код
    [break]
  ...
  [default:
    код]
}
```

Пример:

```
$x=2;
switch ($x) {
  case 1:
    echo ”x равно 1”;
  case 2:
    echo ”x равно 2”;
  case 3:
    echo ”x равно 3”;
}
```

В приведенном выше примере сработают 2-й и 3-й варианты. Если мы хотим, чтобы сработал только один какой-нибудь вариант (только тот, который соответствует значению выражения), то нужно использовать оператор `break`.

Пример:

```
$x=2;
switch ($x) {
  case 1:
    echo ”x равно 1”;
    break;
  case 2:
    echo ”x равно 2”;
    break;
  case 3:
    echo ”x равно 3”;
    break;
}
```

В этом примере сработает только 2-й вариант.

Оператор условия ?:

Оператор условия является сокращенной формой оператора условного перехода `if...else...`. Обычно он применяется вместе с оператором присваивания одного

из двух возможных значений, в зависимости от значения условного выражения. Синтаксис оператора условия следующий:

```
условие ? выражение1 : выражение2
```

С оператором присваивания оператор условия имеет такой вид:

```
переменная=условие ? выражение1 : выражение2
```

Оператор условия возвращает значение выражения *выражение1*, если *условие* истинно, в противном случае — значение выражения *выражение2*. Поэтому он может использоваться в составных выражениях.

Пример:

```
$x=5;
$x<10 ? "x меньше 10": "x больше или равно 10";
echo $x;
```

2.5.9. Операторы цикла

Оператор цикла обеспечивает многократное выполнение блока программного кода до тех пор, пока не выполнится некоторое условие. В PHP предусмотрены четыре оператора цикла: `for`, `while`, `do-while` и `foreach`. Операторы for, while и do-while мы рассмотрим в данном разделе. Оператор foreach (для каждого) специально предназначен для работы с массивами в PHP. Он очень удобен и будет подробно рассмотрен в разд. 2.90.4 (“Перемещение по массиву”).

Оператор цикла обеспечивает многократное выполнение блока программного кода до тех пор, пока не выполнится некоторое условие.

Оператор *for*

Оператор `for` (для) также называют *оператором со счетчиком циклов*, хотя в нем совсем не обязательно использовать счетчик. Синтаксис этого оператора следующий:

```
for ([начальное_выражение]; [условие]; [выражение_обновления])
{
  код
}
```

Здесь квадратные скобки лишь указывают на то, что заключенные в них параметры не являются обязательными. Как и в случае оператора условного перехода, возможна и такая запись:

```
for ([начальное_выражение]; [условие]; [выражение_обновления]) {
  код
}
```

Все, что находится в круглых скобках справа от ключевого слова `for`, называется *заголовком* оператора цикла, а содержимое фигурных скобок — его *телом*.

В заголовке оператора цикла начальное выражение выполняется только один раз в начале выполнения оператора. Второй параметр представляет собой условие продолжения работы оператора цикла. Он аналогичен условию в операторе `if`. Третий параметр содержит выражение, которое выполняется после выполнения всех выражений кода, заключенного в фигурные скобки.

Оператор цикла работает следующим образом. Сначала выполняется *начальное_выражение*. Затем проверяется *условие*. Если оно ложно, то оператор цикла прекращает работу (при этом *код* не выполняется). В противном случае выполняется *код*, расположенный в теле оператора `for`, т. е. между фигурными скобками. После этого выполняется *выражение_обновления* (третий параметр оператора `for`). Так заканчивается первый цикл или, как еще говорят, первая *итерация* цикла. Далее, снова проверяется *условие*, и все повторяется описанным выше способом.

Обычно в качестве начального выражения используют оператор присваивания значения переменной, например, `$i=0`. Имя переменной и присваиваемое значение могут быть любыми. Эту переменную называют *счетчиком* цикла. В этом случае условие, как правило, представляет собой элементарное выражение сравнения переменной счетчика цикла с некоторым числом, например, `$i<=$nMax`. Выражение обновления в таком случае просто изменяет значение счетчика, например, `$i=$i+1` или, короче, `$i++`.

В следующем примере оператор цикла просто изменяет значение своего счетчика, выполняя 15 итераций:

```
for ($i=1; $i <= 15; $i++) {}
```

Немного модифицируем этот код, чтобы вычислить сумму всех целых положительных чисел от 1 до 15:

```
$s=1;
for ($i=1; $i <= 15; $i++) {
  $s=$s + $i;
}
```

Заметим, что счетчик цикла может быть не только возрастающим, но и убывающим.

❒ Пример: `x` в степени `y`.

Допустим, требуется вычислить `x` в степени `y`, где `x`, `y` — целые положительные числа. Алгоритм решения этой задачи прост: надо вычислить выражение `x*x*x*...*x`, в котором сомножитель `x` встречается `y` раз. Очевидно, что если `y` не превышает 10, то можно воспользоваться оператором умножения: выражение будет не очень длинным. А как быть в том случае, когда `y` равно нескольким десяткам или сотням? Ясно, что в общем случае следует применить оператор цикла:

```
/* Вычисляем x в степени y */
$z=$x;     // $z хранит результат
for ($i=2; $i <= $y; $i++) {
```

```
  z=z*x;
}
```

В этом примере результат сохраняется в переменной $z. Ее начальное значение равно $x. Если $y равно 1, то оператор цикла не будет выполняться, поскольку не выполняется условие 2<=1 (обратите внимание на начальное значение счетчика цикла) и, следовательно, мы получим верный результат: $x в степени 1 равно $x. Если $y равно 2, то оператор цикла выполнит одну итерацию, вычислив выражение $z=$z*$x при $z, равном $x (т. е. $z=$x*$x). При $y, равном 3, оператор цикла сделает две итерации. На второй, последней, итерации выражение $z=$z*$x вычисляется при текущем значении $z, равном $x*$x, и, следовательно, становится равным $x*$x*$x (т. е. x в степени 3).

- Пример: n!.

Рассмотрим довольно традиционный при изучении операторов цикла пример вычисления факториала числа. Факториал числа n в математике обозначают как n!. Для n, равного 0 и 1, n! равен 1. В остальных случаях n! равен 2*3*4*...*n. Поскольку возможны два варианта исходных данных, нам потребуется использовать оператор условного перехода. Код, решающий эту задачу, выглядит следующим образом:

```
/* Вычисляем n! */
$z=1;     // z хранит результат n!
if ($n >1) {
  for ($i=2; $i <= $n; $i++) {
    $z=$z*$i;
  }
}
```

Для принудительного (т. е. не по условию) выхода из цикла используется оператор break (прерывание). Если вычислительный процесс встречает этот оператор в теле оператора цикла, то он сразу же завершается без выполнения последующих выражений кода в теле и даже выражения обновления. Обычно оператор break применяется при проверке некоторого дополнительного условия, выполнение которого требует завершения цикла, не смотря на то, что условие в заголовке цикла еще не выполнено. Типовая структура оператора цикла с использованием break имеет следующий вид:

```
for ([начальное_выражение]; [условие1]; [выражение_обновления])
{
  код
  if (условие2){
    код
  break
  }
  код
}
```

Для управления вычислениями в операторе цикла можно также использовать оператор `continue` (продолжение). Так же, как и `break`, этот оператор применяется в теле оператора цикла вместе с оператором условного перехода. Однако в отличие от `break`, оператор `continue` прекращает выполнение последующего кода, выполняет выражение обновления и возвращает вычислительный процесс в начало оператора цикла, где производится проверка условия, указанного в заголовке. Типовая структура оператора цикла с использованием `continue` имеет следующий вид:

```
for ([начальное_выражение]; [условие1]; [выражение_обновления] )
{
  код
  if (условие2){
    код
    continue
  }
  код
}
```

Оператор *while*

Оператор цикла `while` (до тех пор, пока) имеет структуру, более простую, чем оператор `for`, и работает несколько иначе. Синтаксис этого оператора следующий:

```
while (условие)
{
  код
}
```

При выполнении этого оператора сначала производится проверка условия, указанного в заголовке, т. е. в круглых скобках справа от ключевого слова `while`. Если оно истинно, то выполняется код в теле оператора цикла, заключенного в фигурные скобки. В противном случае код не выполняется. При выполнении кода (завершении первой итерации) вычислительный процесс возвращается к заголовку, где снова проверяется условие, и т. д.

Если сравнивать оператор `while` с оператором `for`, то особенность первого заключается в том, что выражение обновления записывается в теле оператора, а не в заголовке. Часто забывают указать это выражение, и в результате цикл не завершается (программа "зависает").

Рассмотрим несколько примеров решения задач, которые мы уже решали ранее с использованием оператора цикла `for`.

❒ Пример: `x` в степени `y`.

```
/* Вычисляем x в степени y */
$z=$x;     // $z хранит результат
$i=2;
while ($i==2) {
```

```
  $z=$z*$x;
  $i++;
}
```

❒ Пример: `n!`.

```
/* Ввычисляем n! */
$z=1;     // $z хранит результат $n!
if ($n >1) {
  $i=2;
  while ($i <= $n ) {
    $z=$z*$i;
    $i++;
  }
}
```

Во всех приведенных выше примерах использовался счетчик цикла. Однако он инициировался заранее, до оператора цикла. Обновление значения этого счетчика производилось в теле оператора цикла.

Для управления вычислительным процессом в операторе `while`, так же как и в операторе `for`, можно применять операторы прерывания `break` и продолжения `continue`.

> ***Внимание***
> Если в `while` вы используете счетчики циклов, то будьте осторожны, применяя `break` и `continue`.

Оператор *do-while*

Оператор `do-while` (делай до тех пор, пока) представляет собой конструкцию из двух операторов, используемых совместно. Синтаксис этой конструкции следующий:

```
do {
  код
}
while (условие);
```

В отличие от оператора `while` в операторе `do-while` код выполняется хотя бы один раз, независимо от условия. Условие проверяется после выполнения кода. Если оно истинно, то снова выполняется код в теле оператора `do`. В противном случае работа оператора `do-while` завершается. В операторе `while` условие проверяется в первую очередь, до выполнения кода в теле. Если при первом обращении к оператору `while` условие ложно, то код не будет выполнен никогда.

Рассмотрим несколько примеров решения задач, которые мы уже решали ранее с использованием операторов цикла `for` и `while`.

❒ Пример: `x` в степени `y`.

```
/* Вычисляем x в степени y */
$z=$x;     // $z хранит результат
```

```
$i=2;
do{
  $z=$z*$x;
  $i++;
}
while ($i <= $y);
```

- Пример: `n!`.

```
/* Вычисляем n! */
$z=1;     // $z хранит результат $n!
if ($n >1) {
  $i=2;
  do {
    $z=$z*$i;
    $i++;
  }
  while ( $i <= $n );
}
```

2.6. Строки

Работа со строками в PHP является очень важной, особенно при решении задач, связанных с Web-сайтами. Строковые значения (последовательности символов) должны быть заключены в двойные или одинарные кавычки, а от вида кавычек зависит, как строка будет отображена.

2.6.1. Двойные и одинарные кавычки

В двойных кавычках помимо обычных видимых и служебных символов могут находиться имена переменных (идентификаторы, начинающиеся с символа $), а также теги HTML. При выводе таких строк на экран с помощью оператора echo интерпретатор PHP заменяет идентификаторы переменных их значениями, а теги HTML интерпретируются браузером.

Переменные в строках, заключенных в одинарные кавычки, при выводе с помощью оператора echo не заменяются значениями. Однако теги HTML интерпретируются так же, как и в случае двойных кавычек.

> ***Внимание!***
> При выводе строк оператором echo служебные символы, такие как \n (перевод строки), \r (возврат каретки) и \t (табуляция), в браузере не выполняют своей функции, но отображаются в виде пробела. Это не происходит, если строковое значение заключено в одинарные кавычки. Служебные символы в строках, заключенных в двойные

кавычки, выполняют форматирующую роль при использовании некоторых специальных функций (см. разд. 2.6.4), а также PHP CLI.

На рис. 2.9 показано, как отображаются в браузере строки со служебными символами.

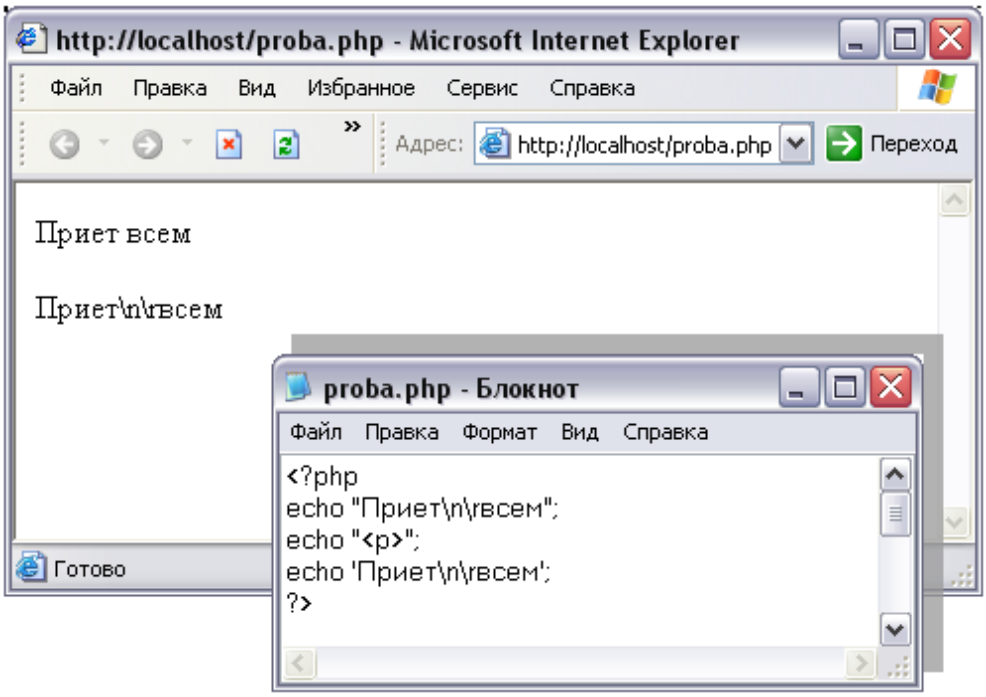

Рис. 2.9. Отображение служебных символов в браузере в зависимости от вида кавычек

Длинные последовательности символов, которые в текстовом редакторе занимают несколько строк, можно задавать с помощью так называемого heredoc-синтаксиса, не используя кавычек:

Переменная=<<<*МЕТКА*

текст

МЕТКА;

Здесь *МЕТКА* — произвольная последовательность букв, цифр и символа подчеркивания, начинающаяся не с цифры.

Пример:

```
<?php
```

```
$x=<<<MYLABEL
       Во первых
       строках
       своего письма
MYLABEL;
echo $x;
?>
```

При работе с модулем PHP CLI (с интерфейсом командной строки) текст на экране будет иметь такое же расположение, что и между метками в PHP-коде. Однако в окне браузера он будет отображен в виде одной или нескольких строк в зависимости от ширины клиентской области браузера. При этом части текста, расположенные в PHP-коде в различных строках могут отображаться в одной строке через пробел.

Рассмотренный выше пример в окне браузера отображается точно так же, как и следующий код:

```
<?php
$x="
       Во первых
       строках
       своего письма
";
echo $x;
?>
```

Внимание

При использовании heredoc-синтаксиса необходимо соблюдать следующие правила:

За первым идентификатором метки не должно быть никаких символов, в том числе и пробелов.

За заключительным идентификатором метки должен находиться единственный символ точки с запятой.

Чтобы в строке, заключенной в двойные кавычки, отключить интерпретацию таких служебных символов, как двойные кавычки и $, перед ними следует указать обратный слэш (\). Фигурные скобки не отображаются, если перед открывающей скобкой не поставить обратный слэш.

Строки в одинарных кавычках хранятся как есть за исключением пары \', которая хранится как символ апострофа (одинарная кавычка)

На рис. 2.10 показаны примеры действия обратного слэша, а также сочетаний кавычек различного вида.

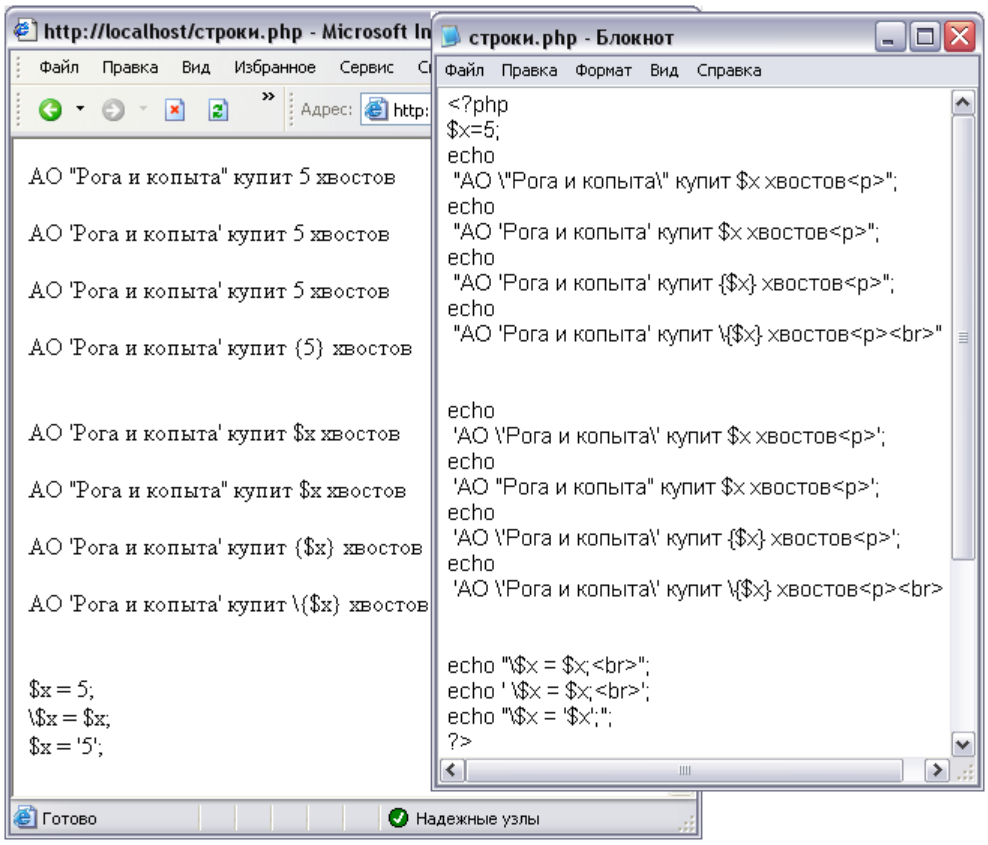

Рис. 2.10. Действие обратного слэша в строках

Как уже отмечалось ранее, в строках могут встречаться теги HTML, которые позволяют отформатировать выводимый текст, а также выполнить другие действия. Интерпретатор PHP не интерпретирует их, а сохраняет как есть. С помощью оператора echo они направляются в выходной документ, который отображается браузером. Поскольку внутри HTML-кода могут использоваться кавычки различных видов, при формировании строк в PHP следует быть внимательным, чтобы правильно их расставлять и сочетать друг с другом.

2.6.2. Склейка строк

Соединение двух строк производится посредством операции склейки (конкатенация), которая обозначается точкой (.). В результате строка справа от этого символа приписывается без пробелов к концу строки, расположенной слева.

Пример:

```
$x="Привет";
$y=$x." всем";                      // Привет всем
$y="$x всем"." моим друзьям";       // Привет всем моим друзьям
$y.=" и знакомым";                  // Привет всем моим друзьям и знакомым
```

В операции склейки могут участвовать не только строки, но и данные других типов. При этом все операнды приводятся к строковому типу автоматически.

Пример:

```
$x=5;
$y=$x." рублей";      // 5 рублей
$y=($x/2)." рубля";   // 2.5 рубля
$z=3>2;               // true
$y=$x.$z;             // 51
```

Обратите внимание на применение круглых скобок в третьем выражении, задающих порядок вычислений. Без них было бы выведено сообщение об ошибке.

2.6.3. Преобразование строк

Для преобразования символьных строк в PHP имеется большое количество встроенных функций. Здесь мы рассмотрим лишь несколько наиболее важных из них:

- strlen(*строка*) — возвращает длину строки (количество символов), указанной в качестве параметра. При этом служебные символы, такие как перевод строки, табуляция и др. также будут подсчитаны. Например, строка "Привет, друзья!" может быть набрана в PHP-коде по-разному, что может сказаться на ее длине, хотя все варианты при этом могут отображаться одинаково:

  $x="Привет, друзья!";

  strlen($x); // возвращает 15

  $x="Привет,

  друзья"; // возвращает 16

  Здесь во втором варианте присваивания значения переменной $x после запятой присутствует невидимый служебный символ перевода строки, который в действительности состоит из двух символов, а перед словом "друзья нет ни одного пробела" (в первом варианте есть один пробел). Не трудно подсчитать, что 15-1+2=16. Данное обстоятельство следует учесть, если вам потребуется написать какой-нибудь код анализа текстовых строк.

- trim(*строка*), ltrim(*строка*), rtrim(*строка*) — удаляют ведущие и заключительные пробелы, только ведущие и только заключительные пробелы соответственно.

- strchr(*строка*, *символ*) — возвращает часть строки, начиная с указанного символа и до конца строки. Если строка не найдена, то возвращается false.

  Пример:

strchr(“BcADaf ag”, “a”); // af ag

- stristr(*строка*, *символ*) — возвращает часть строки, начиная с указанного символа и до конца строки. Если строка не найдена, то возвращается false. В отличие от strchr() регистр строки и символа не учитывается.

  Пример:

  strchar(“BcADaf ag”, “a”); // ADaf ag

- strichr(*строка*, *символ*) — возвращает часть строки, начиная с указанного символа и до конца строки. Если строка не найдена, то возвращается false. В отличие от strchr() поиск символа ведется от конца строки.

  Пример:

  strrchr(“BcADaf ag”, “a”); // ag

- strpos(*строка1*, *строка2*) — возвращает позицию первого вхождения второй строки в первую. Если строка не найдена, то возвращается false. Нумерация позиций начинается с нуля и производится слева направо.

  Пример:

  strpos(“Привет, мои друзья и мои знакомые”, “мои”); // 8

- strrpos(*строка1*, *строка2*) — возвращает позицию последнего вхождения второй строки в первую. Если строка не найдена, то возвращается false. Нумерация позиций начинается с нуля и производится слева направо.

  Пример:

  strrpos(“Привет, мои друзья и мои знакомые”, “мои”); // 21

- substr(*строка*, *число1*, *число2*) — возвращает подстроку, начинающуюся с позиции число1 и длиной число2. Нумерация позиций начинается с нуля и производится слева направо.

  Пример:

  substr(“Привет, мои друзья и мои знакомые”, 8, 10); // мои друзья

- str_replace(*строка1*, *строка2*, *строка*) — заменяет в *строка* все вхождения *строка1*на *строка2*. В качестве заменяемой и замещающих строк могут быть не только видимые символы, но и символы табуляции, конца строки и файла

  Пример:

  str_replace(“мои”, “наши”, “Привет, мои друзья и мои знакомые”);

  /* Привет, наши друзья и наши знакомые */

- strtr(*строка*, *строка1*, *строка2*) — заменяет в *строка* каждое вхождение любого символа из *строка1*на соответствующий ему символ в *строка2*. Если *строка1*и *строка2* имеют различные длины, то “лишние” символы в более длинной строке не используются.

  Пример:

  strtr("Привет, мои друзья", "иуяПрвтдзь","iuaPrvtdzj"); // Privet, moi druzja

Здесь второй и третий параметры функции содержат информацию о соответствии символов для “перекодировки” строки, указанной в качестве первого параметра. Так, в примере каждое вхождение буквы “и” заменяется буквой “i”, “у” — “u” и т.д.

- str_repeat(*строка, число*) — повторяет указанную строку заданное количество раз и возвращает ее.

  Пример:

  str_repeat(“#”,10); // ##########

- strrev(*строка*) — возвращает указанную строку, заменив порядок следования символов на противоположный.

  Пример:

  strrev(“абвгд”); // дгвба

- strtolower(*строка*) — возвращает строку, преобразовав все символы указанной строки в нижний регистр.
- strtoupper(*строка*) — возвращает строку, преобразовав все символы указанной строки в верхний регистр.
- ucfirst(*строка*) — преобразует первый символ указанной строки в верхний регистр и возвращает ее.
- ucwords(*строка*) — преобразует первый символ каждого слова указанной строки в верхний регистр и возвращает ее.
- str_word_count(*строка* [, *формат*]) — возвращает массив слов, входящих в строку или количество слов в строке, если второй параметр не указан. Параметр *формат* может принимать два целочисленных значения, которые определяют вид возвращаемого массива:
  - 1 — числовые индексы элементов массива соответствуют порядковому номеру слова в строке; нумерация начинается с 0.
  - 2 — числовые индексы элементов массива соответствуют позиции слова в строке; нумерация начинается с 0.

  Пример:

  $x="How do you do";

  str_word_count ($x); // 4

  print_r(str_word_count ($x, 1));

  /* Выводит информацию о массиве слов: */

  Array (

  [0] => How

  [1] => do

  [2] => you

```
[3] => do
 )
print_r(str_word_count ($x, 2);
/* Выводит информацию о массиве слов: */
Array (
[0] => How
[4] => do
[7] => you
[11] => do
 )
```

Примечание

Массивы подробно рассматриваются в разд. 2.10.

- strtok(*строка1*, *строка2*) — возвращает первую подстроку строки *строка1*, отделенную от остальной части любым из символов-разделителей, указанных в строке *строка1*. Если исходная строка не содержит указанных разделителей, то возвращается эта строка целиком.

  Пример:

```
<?php
$string = "Это пример\nстроки";
$word=strtok($string, " \n");     // значение $word равно "Это"
/?>
```

  Здесь во втором параметре функции strtok() указано два разделителя — пробел и перевод строки (\n). Разумеется, вы можете указать там и другие символы, например, символ табуляции (\t), вертикальную черту (|), запятую (,) и др. специальные или обычные символы.

  Данная функция имеет особенности, которые следует учесть при ее применении. А именно, strtok() сохраняет позицию выделенной подстроки и при втором и последующих вызовах ей достаточно передать в качестве единственного параметра только строку разделителей, чтобы разбить всю исходную строку на подстроки. В следующем примере выводятся все слова (подстроки), которые принципиально можно выделить посредством таких разделителей, как пробел, перевод строки (\n) и табуляция (\t). Обратите внимание, что в исходной строке символ табуляции отсутствует, а потому этот символ во втором параметре функции не производит никакого действия.

  Пример:

```
<?php
$string = "Это пример\nстроки";
```

```
$word = strtok($string, " \n\t");
while ($word) {
   echo "$word<br>";
   $word = strtok(" \n\t");
}
?>
```

В результате выполнения этого кода, на экран будет выведен список из трех слов, которые выделил из исходной строки оператор echo. Заметьте, что тег “
”, добавляемый к выводимой строке, обеспечивает лишь вывод обнаруженных слов в отдельных строках. Иначе говоря, он здесь нужен лишь для форматирования вывода.

Начиная с версии PHP 2.1.0, пустые подстроки пропускаются функцией strtok().

Итак, функция strtok() берет в качестве прараметра некоторую строку и возвращает подстроку в соответствии с заданными разделителями. Повторными вызовами, обычно в цикле (операторы for, while, do-while), с помощью этой функции можно “выудить” все подобные подстроки.

На практиее обычно требуется преобразовать строку в массив или наоборот. В PHP для этого используются функции explode() и implode(), которые мы рассмотрим в разд. 2.9.

2.6.4. Форматирование строк

Для форматирования текстовых строк, выводимых в окно браузера оператором echo, часто используется HTML и каскадные таблицы стилей. Служебные символы \n (перевод строки) и \t (табуляция) также можно использовать в строках. Если они находятся в строке, заключенной в одинарные кавычки, то при выводе этой строки оператором echo эти символы не будут работать по своему назначению и будут видны на экране. В случае двойных кавычек они, вообще говоря, также не сработают, но отобразятся в виде пробелов. Однако если вы хотите все же заставить их функционировать, то строку в двойных кавычках следует начать тегом <pre>. Например, следующий оператор выведет сообщение в виде двух строк, причем название фирмы отобразится жирным шрифтом:

```
echo “<pre><b>АО ‘Рога и копыта’<b>\nприглашает на работу главного бухгалтера”
```

Вместе с тем, с целью форматирования можно применять специальные функции PHP printf() и sprintf(). Если вы используете PHP CLI, работающий в режиме командной строки, то специальные функции являются единственным средством форматирования строк.

На практике часто приходится формировать строки, содержащие не только обычные алфавитные символы и знаки препинания, но и числа и значения переменных. При этом числа можно представлять в различных системах счисления, а в десятичной

системе — в разичных видах. В функциях printf() и sprintf() можно указать шаблон, в соответстии с которым будет отформатирована строка. Эти функции имеют одинаковый синтаксис, но отличаются действием и возвращаемым значением:

- printf(*формат_строка*, *список_переменных*) — выводит отформатированную строку и возвращает длину этой строки;
- sprintf(*формат_строка*, *список_переменных*) — возвращает отформатированную строку (но не выводит ее).

Список переменных — имена переменных, разделенные запятой, значения которых следует подставить в форматируемую строку. Этот параметр не обязателен.

Параметр *формат_строка* представляет собой текстовую строку, которую требуется отформатировать в соответствии с описателями преобразований. Описатели преобразований находятся в форматируемой строке и каждый из них описывает представление значения соответствующей переменной, указанной в списке: первый описатель для первой переменной, второй — для второй переменной и т.д. Разумеется, строка может и не содержать описателей. В этом случае она просто не будет отформатирована.

Каждый описатель прреобразований состоит из знака процента (%), за которым следует один или более дополнительных элементов в следующемтом порядке: %*ЗаполнительВыравниваниеДлинаТочностьТип*. Рассмотрим эти необязательные элементы.

1. Заполнитель — символ, который будет использоваться для дополнения результата преобразования до заданной длины (см. далее). Это может быть пробел (по умолчанию), *0* или любой другой символ, перед которым необходимо указать одинарную кавычку (‘).
2. Выравнивание — символ минус (-); если он указан, то выравнивание происходит по левому краю поля, выделенного для отображения значения, а в противном случае (т.е. по умолчанию) — по правому краю.
3. Длина — число, определяющее ширину поля, отводимого для вывода результата преобразования. Если результат содержит меньше символов, то оставшееся пространство будет заполнено пробелом или символом заполнения, если он был указан.
4. Точность — число, перед которым указывается точка. Определяет, сколько десятичных разрядов в дробной части числа следует отображать. Если число имеет больше таких разрядов, то результат получается округлением, а не отбрасыванием лишних знаков. Применяется для чисел с плавающей точкой (float).
5. Тип — символ, определяющий, как трактовать тип значения переменной из списка, которое подставляется в форматируемую строку. Допустимые следующие указатели типа:
   - % — значение переменной не используется
   - *b* — значение трактуется как целое и выводится в виде двоичного числа

- c — значение трактуется как целое и выводится в виде символа с соответствующим кодом ASCII
- d — значение трактуется как целое и выводится в виде десятичного числа со знаком
- e — значение трактуется как число с плавающей точкой и представляется в экспоненциальной форме
- *u* - значение трактуется как целое и выводится в виде десятичного числа без знака
- *f* — значение трактуется как число с плавающей точкой и представляется в неэкспоненциальной форме (т.е. только с разделительной точкой)
- *o* — значение трактуется как целое и выводится в виде восьмеричного числа
- *s* — значение трактуется как строка
- *x* — значение трактуется как целое и выводится в виде шестнадцатиричного числа в нижнем регистре
- *X* — значение трактуется как целое и выводится в виде шестнадцатиричного числа в верхнем регистре

В листинге 2.3. приведен код, в котором используется функция printf() для форматирования строк так, чтобы получился список. Результат выполнения этого кода показан на рис. 2.11.

Листинг 2.3. Пример форматирования строк (вариант 1)

```
<?php
$x1=2700;
$x2=14.378;
$x3=145.2;
$n1="Бендер";
$n2="Балаганов";
$n3="Паниковский";
$format="%'.-12s%'.10.2f руб.\n";
printf("<pre><b>Платежная ведомость</b>\n");
printf($format, $n1, $x1);
printf($format, $n2, $x2);
printf($format, $n3, $x3);
?>
```

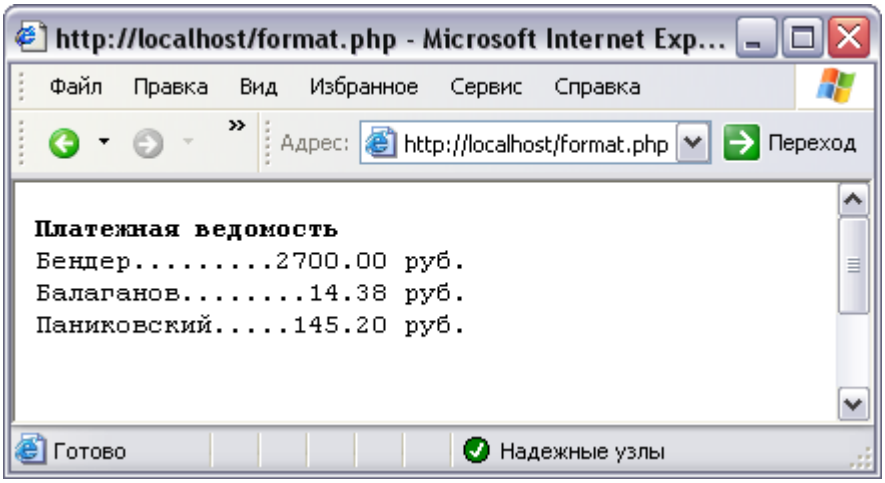

Рис. 2.11. Пример форматирования строк

В данном примере был использован тег <pre>, который обеспечивает отображение браузером строк в соответствии с их предварительным форматированием. Здесь это понадобилось, чтобы сработали служебные символы \n для перехода на другую строку. В качестве заполнителя была применена точка (‘.). Фамилии выравнивались по левому краю (-) поля шириной 12 символов с заполнителем в виде точки (‘.). Таким образом, описатель преобразования для строковых значений имеет вид: %’.-12s.

Числа представлялись с десятичной точкой и двумя разрядами после нее. Они выравнивались по правому краю поля из 10 символов, а в качестве заполнителя использовалась точка. Таким образом, описатель преобразования для чисел имеет вид: %’.10.2f.

Такой же результат можно получить, используя функцию sprintf() для создания отформатированной строки и оператор echo — для ее вывода (листинг 2.4).

Листинг 2.4. Пример форматирования строк (вариант 2)

```
<?php
$x1=2700;
$x2=14.378;
$x3=145.2;
$n1="Бендер";
$n2="Балаганов";
$n3="Паниковский";
$format="%'.-12s%'.10.2f руб.\n";
echo
"<pre><b>Платежная ведомость</b>\n".
sprintf($format,$n1, $x1).
```

```
sprintf($format,$n2, $x2).
sprintf($format,$n3, $x3);
?>
```

Заметим, что порядок описателей в форматируемой строке соответствует порядку других параметров функций sprintf() и printf(). Для PHP версии 2.0.6 и старше можно изменить этот порядок и даже использовать один и тот же описатель несколько раз. Для этого в каждом описателе сразу же за имволом % следует указать номер, обратный слэш и символ $ (далее могут следовать другие элементы). Этот номер соответствует порядковому номеру последующих аргументов функций sprintf() и printf(). В листинге 2.5. приведен пример кода, выводящего строку “Человек, обезьяна и корова есть млекопитающее”.

Листинг 2.5. Пример форматирования с использованием нумерации параметров

```
<?php
$x=”человек”;
$y=”обезьяна”;
$z=”корова”;
$type=”млекопетающее”;
echo
 ucfirst(sprintf(“%2\$s, %3\$s и %4\$s есть %1\$s”, $type, $x, $y, $z));
?>
```

В результате выполнения этого кода будет выведена строка “Человек, обезьяна и корова есть млекопитающее”. Здесь функция ucfirst() использовалась для того, чтобы отформатированная строка начиналась с прописной (заглавной) буквы.

2.7. Числа

В PHP числа могут быть только двух типов: целые (integer) и с плавающей точкой (float). Целые числа не имеют дробной части и не содержат разделительной точки. Числа с плавающей точкой имеют целую и дробную части, разделенные точкой.

Операции с целыми числами процессор компьютера выполняет значительно быстрее, чем операции с числами, имеющими точку. Это обстоятельство имеет смысл учитывать, когда расчетов много. Например, индексы, длины строк являются целочисленными. Число π, многие числа, полученные с помощью оператора деления, денежные суммы и т. п. являются числами с плавающей точкой.

В PHP можно производить операции с числами различных типов. Однако при этом следует знать, какого числового типа будет результат. Если результат операции является числом с дробной частью, то он представляется как число с плавающей точкой. Если результат оказался без дробной части, то он приводится к целочисленному типу, а не представляется числом, у которого в дробной части одни нули.

Отметим еще одно важное обстоятельство: арифметические операторы можно выполнять над строками, содедержащими числа. Другие, более сложные, вычисления можно выполнять с помощью встроенных функций, которых в PHP имеется большое количество. Кроме того, имеются функции, позволяющие отбражать числа в нужном виде.

2.7.1. Математические функции

Из большого набора имеющихся в PHP математических функций приведем лишь наиболее употребительные:

- `abs(`*число*`)` — модуль (абсолютное значение) числа;
- `acos(`*число*`)` — арккосинус числа;
- `asin(`*число*`)` — арксинус числа;
- `atan(`*число*`)` — арктангенс числа;
- `atan2(`*x*`, `*y*`)` — угол в полярных координатах точки;
- `ceil(`*число*`)` — округление числа вверх до ближайшего целого;
- `cos(`*число*`)` — косинус числа;
- `exp(`*число*`)` — число `e` в степени число;
- `floor(`*число*`)` — округление числа вниз до ближайшего целого;
- fmod(*число1*,*число2*) — возвращает дробный остаток от деления первого числа на второе. Этот остаток определяется как число, удовлетворяющее уравнению *число1* = i**число2* + r, где i - некоторое целое. Остаток r имеет такой же знак, что и *число1* .
- `log(`*число*`)` — натуральный логарифм числа;
- `log10(`*число*`)` — десятичный логарифм числа;
- `max(`*число1*`, `*число2*`,…,`*числоN*`)` — большее из чисел в списке;
- `max(массив_чисел)` — большее из чисел в массиве;
- `min(`*число1*`, `*число2*`,…,`*числоN*`)` — меньшее из чисел в списке;
- `max(массив_чисел)` — большее из чисел в массиве;
- `pow(`*число1*`, `*число2*`)` — *число1* в степени *число2*;
- `rand(`*число1*`, `*число2*`)` — случайное число между заданными числами;
- `round(`*число*`)` — округление числа до ближайшего целого;
- `sin(`*число*`)` — синус числа;
- `sqrt(`*число*`)` — квадратный корень из числа;
- srand(*число*) — устанавливает начальное значение генератора случайных чисел в указанное число; если параметр не указан, то начальное число выбирается случайно.

❒ `tan(`*число*`)` — тангенс числа.

2.7.2. Математические константы

В PHP имеется множество предопределенных числовых констант, которые используются в инженерных и научных расчетах (табл. 2.8).

Таблица2.8.Математические константы

Имя константы	**Значение**	**Описние**
M_PI	3.14159265358979323846	Число π
M_E	2.7182818284590452354	Число e (основание натуральных логарифмов)
M_LOG2E	1.4426950408889634074	$\log_2 e$
M_LOG10E	0.43429448190325182765	lg e
M_LN2	0.69314718055994530942	ln e
M_LN10	2.30258509299404568402	ln 10
M_PI_2	1.57079632679489661923	$\pi/2$
M_PI_4	0.78539816339744830962	$\pi/4$
M_1_PI	0.31830988618379067154	$1/\pi$
M_2_PI	0.63661977236758134308	$2/\pi$
M_SQRTPI	1.77245385090551602729	$\sqrt{\pi}$
M_2_PI	0.63661977236758134308	$2/\sqrt{\pi}$
M_SQRTPI	1.77245385090551602729	$\sqrt{2}$
M_SQRT3	1.73205080756887729352	$\sqrt{3}$
M_SQRT1_2	0.70710678118654752440	$1/\sqrt{2}$
M_LNPI	1.14472988584940017414	$\ln \pi$
M_EULER	0.57721566490153286061	Постоянная Эйлера

2.7.3. Представление чисел в различных системах счисления

Числа в PHP можно представлять в различных системах счисления, т. е. в системах с различными основаниями: 10 (десятичной), 16 (шестнадцатеричной), 8 (восьмеричной) и 2 (двоичной). Десятичная форма представления чисел,

использующая цифры от 0 до 9, наиболее нам привычна в обиходе. Однако в компьютерных технологиях часто используются и другие системы: восьмеричная (цифры от 0 до 7), шестадцатеричная (цифры 0, …, 9, a, b, …, f) и двоичная (цифры 0 и 1). Например, десятичное число 17 в восьмеричной системе представляется как 15, в шестандатеричной — как b0, а в двоичной — как 10001.

Для преобразований чисел из одной системы счисления в другую служат специальные функции:

- base_convert (*строка1, основание1, основание2*) — преобразует строку *строка1,* содержащую число в системе счисления по основанию *основание1,* в строку с числом по основанию *основание2* и возвращает строку, содержащую результат этого преобразования.

  Примеры:

  base_convert(“17”, 10, 16); // “11”

  base_convert(“17”, 10, 8); // “21”

  base_convert(“17”, 10, 2); // “111”

  base_convert(“ff”, 16, 10); // “255”

  base_convert(“101”, 2, 10); // “5”

  Обратите внимание, что данная функция возвращает строку, содержащую число в том или ином представлении, а не число (т.е. данные числового типа). При рассмотрении следующих функций обратите внимание на то, кого типа данные они возвращают и принимают в качестве параметров..

- dechex(*целое_число*)— возвращает строку, содержащую указанное целое десятичное число, в шестнадцатеричном виде.

  Примеры:

  dechex(3); // “3”

  dechex(250); // “fa”

  dechex(255); // “ff”

- hexdec*(строка*) — возвращает десятичное число, соответствующее указанному в строковом параметре шестнадцатеричному числу.

  Пример:

  hexdec(“a”); // 10

- decbin(*целое_число*)— возвращает строку, содержащую указанное целое десятичное число, в двоичном виде.

  Примеры:

  decbin(3); // “11”

  decbin(250); // “11111010”

  decbin(255); // “11111111”

- bindec(*строка*) — возвращает десятичное число, соответствующее указанному в строковом параметре двоичному числу.

  Пример:

  hexdec(“101”); // 5

- decoct(*целое_число*)— возвращает строку, содержащую указанное целое десятичное число, в восьмеричном виде.

  Примеры:

  decoct(3); // “3”

  decoct(250); // “372”

  decoct(255); // “377”

- octdec((*строка*) — возвращает десятичное число, соответствующее указанному в строковом параметре восьмеричному числу.

  Пример:

  octdec(“10”); //7

> ***Внимание***
> Во всех перечисленных выше функциях строковые параметры можно передавать и без кавычек (т.е. в виде литералов).

Пдставления чисел в различных системах счисления, отличных от десятичной, являются строковыми данными. Арифметические операторы для этих данных действуют так, как будто это десятичные числа, не зависимо от того, принадлежат они к числовому или строковому типу. Иначе говоря, арифметика остается десятичной, не зависимо от системы счисления операндов.

Примеры:

decbin(2)+decbin(3); // 21 (“10”+”11”)

dechex(3)+dechex(16); // 13 (“3”+”10”)

bindec("10")+bindec("11"); // 5 (2+3)

2.7.4. Форматирование чисел

Числа в PHP хранятся и выводятся в наиболее эффективном формате. Например, если переменная имеет значение 15.00, оно будет отображено на экране как 15. В приложениях может потребоваться отобразить число в некотором определенном формате, например, разделить группы разрядов пробелом, отобразить два знака после разделительной точки для представления денежных сумм и т.п.. Некоторые возможности форматирования чисел мы уже рассматривали в разд. 2.6.4. — функции printf() и sprintf(). Здесь же мы остановимся на специальной функции форматирования чисел:

number_format(*число*, *точность*, *разделитель1*, *разделитель2*)

Эта функция возвращает строку, содержащую отформатированное число, и принимает следующие параметры:

- *число* — число (float), которое требуется отформатировать; если следующие параметры не используются, то число будет представлено без дробной части (с округлением до ближайшего большего целого) и запятыми в качестве разделителя групп по три разряда.
- *точность* — целое число, указывающее, сколько знаков в дробной части числа следует показать (по умолчанию — 0); этот параметр обязателен, если вы собираетесь использовать параметры *разделитель1* и *разделитель2*; если вы указываете меньшее количество знаков, чем есть в числе, то происходит округление в большую сторону;
- *разделитель1* — строка с символом, который используется для разделения целой и дробной частей числа (по умолчанию — точка); этот параметр обязателен, если вы собираетесь использовать параметр *разделитель2*;
- *разделитель2* — строка с символом, который используется для разделения групп по три разряда в целой части числа (по умолчанию — запятая); этот параметр обязателен, коль скоро вы использовали *разделитель1*.

Таким образом, функция number_format() может принимать один, два или четыре параметра, но не три. Третий и четвертый параметры могут быть строками из нескольких символов, но качестве разделителей будут использованы лишь первые их символы.

На рис. 2.12 приведены примеры использования форматирования чисел.

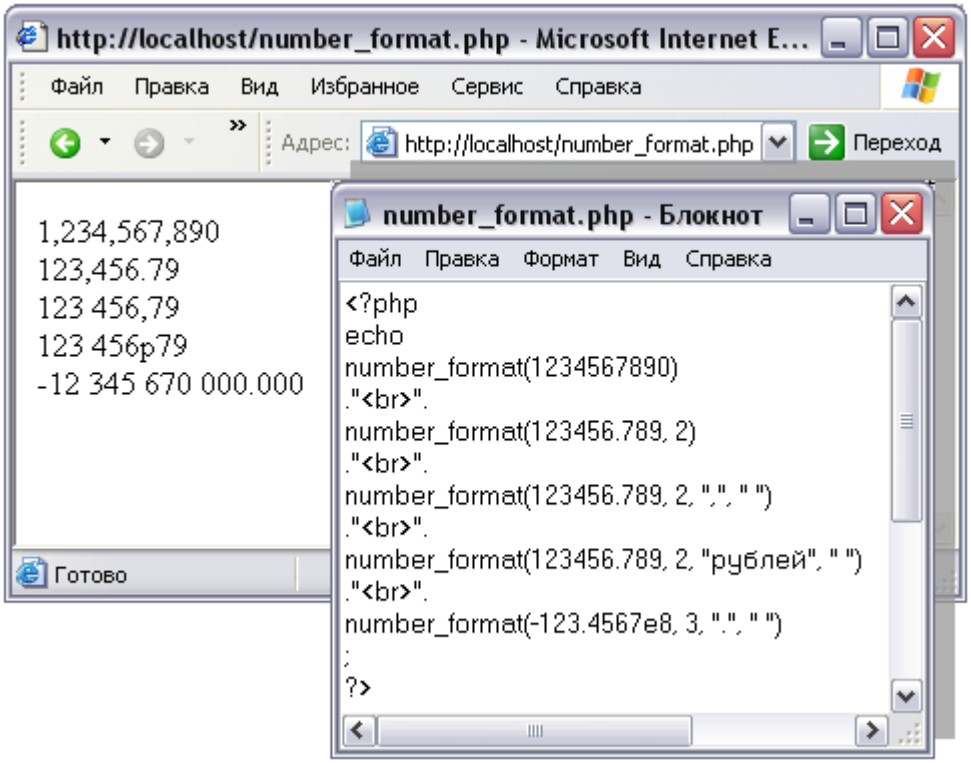

Рис. 2.12. Примеры форматирования чисел с помощью функции number_format()

Примечание

В западной традиции в качестве разделителя целой и дробной частей числа принято использовать точку, а для разделения групп разрядов — запятую. В отечественных документах применяют запятую и пробел соответственно.

Внимание

Математические вычисления следует выполнять до форматирования чисел. Не забывайте, что функция форматирования возвращает текстовую строку и применяется для вывода чисел в надлежащем виде.

2.8. Дата и время

В основе всех операций, связанных с датами и временем, лежат текущие системные дата и время, установленные на вашем компьютере. Со временем дела обстоят не так просто, как кажется на первый взгляд. Вспомните, что существуют временны́е зоны (часовые пояса), а также сезонные поправки времени. Чтобы иметь возможность координировать деятельность во времени организаций и физических лиц в различных точках нашей планеты, была введена система отсчета времени. Она связана с меридианом, проходящим через астрономическую обсерваторию в городе

Гринвич в Великобритании. Эту временную зону называют средним временем по Гринвичу (Greenwich Mean Time, GMT) или всеобщим скоординированным временем (Coordinated Universal Time, UTC).

Если системные часы вашего компьютера установлены правильно, то отсчет времени производится в системе GMT. Однако в Панели управления обычно устанавливается локальное время, соответствующее конкретному часовому поясу. При создании и изменении файлов на вашем компьютере фиксируется именно локальное время. Вместе с тем операционная система знает разницу между локальным временем и GMT. При перемещении компьютера из одного часового пояса в другой, необходимо изменить установки именно часового пояса, а не текущего системного времени (показания системных часов). Даты и время, генерируемые в сценариях, сохраняются в памяти в системе GMT, но пользователю выводятся, как правило, в локальном виде.

Дата и время хранятся в компьютере в специальном формате timestamp как количество секунд, прошедшее от 1.01.1970 00:00:00 GMT (или, что то же самое, — UCT). В приложениях обычно требуется другое представление даты и времени, для чего в PHP предусмотрены специальные функции.

Текущее системное время в формате timestamp (т.е. как целое количество секунд) получается с помощью функции time(). Для получения в этом же формате любого другого времени служит функция

mktime(*час*, *мин*, *сек*, *мес*, *день*, *год*)

с целочисленными параметрами, смысл которых ясен. Например, вызов функции mktime(12, 49, 0, 2, 9, 2006) вернет число 1139478540 — количество секунд, прошедшее от 1.01.1970 00:00:00 до 9.02. 2006 12:49:00 GMT.

Для представления даты и времени в требуемом формате как строки можно использовать функцию

date(*формат*, *timestamp*)

Здесь параметр *timestamp* — целое число, соответствующее количеству секунд, прошедших с момента 1.01.1970 00:00:00 GMT, а *формат* — строка символов форматирования даты и времени. Допустимые символы представлены в табл. 2.9.

Таблица 2.9. Символы форматирования даты и времени

Символ	**Значение**
d	День месяца (две цифры, используется ведущий 0)
j	День месяца (одна или две цифры)
m	Месяц (две цифры, используется ведущий 0)
n	Месяц (одна или две цифры)
Y	Год (четыре цифры)
y	Год (две цифры)

w	Номер дня недели (0 — воскресенье, 6 — суббота)
g	Часы (12-часовой формат без ведущего 0, от 1 до 12)
G	Часы (24-часовой формат без ведущего 0, от 0 до 23)
h	Часы (12-часовой формат с ведущим 0, от 01 до 12)
H	Часы (24-часовой формат с ведущим 0, от 00 до 23)
i	Минуты (от 00 до 59)
s	Секунды (от 00 до 59)
a	До или после полудня: am или pm (в нижнем регистре)
A	До или после полудня: AM или PM (в верхнем регистре)
U	Целое количество секунд, прошедшее от 1.01.1970 00:00:00 GMT
M	Трехбуквенное сокращение анг. названия месяца
F	Англ. название месяца
D	Трехбуквенное сокращение англ. названия дня недели
l	Англ. название дня недели

Между символами форматирования можно использовать символы-разделители, например, дефис (-), точка (.), косая черта (/), двоеточие (:), пробел и др..

Если второй параметр функции date() не указан, то предполагается текущее время.

Примеры:

```
date("d.m.Y");                          // текущая дата, например, 10.02.2006
date("d.m.Y H:i:s", 10000000000);       // 07.09.2014 08:50:08
date("j F.Yг. g:i:s A", 10000000000);   // 7 September.2014г. 8:50:08 AM
```

Чтобы выполнить какие-либо вычисления над датами-временем, следует сначала перевести их в формат timestamp. В следующем примере выводится отформатированная дата-время, соответствующая 12 часам 49 минутам 0 секунд пяти суткам спустя после 9 февраля 2006г.:

```
<?php
$t=mktime(12, 49, 0, 2, 9, 2006) + 5*24*60*60;
echo date("d.m.Y H:i:s", $t);          // 14.02.2006 12:49:00
?>
```

Для перевода текстовых строк, содержащих дату и время, в формат timestamp служит функция

strtotime(*дата-время*, *timestamp*)

Эта функция принимает строку, которая содержащит ключевые слова, связанные с датой и временем и необязательный целочисленный параметр. Если он указан, то функция возвращает количество секунд относительно его значения, а в противном случае — относительно текущего времени. При возникновении ошибки преобразования функция возвращает -1.

В строковом параметре *дата-время* Функция strtotime() распазнает следующие ключевые слова на англ. языке:

- ❒ Названия месяцев и их трехбуквенные сокращения
- ❒ Названия дней недели и их трехбуквенные сокращения
- ❒ Названия елементов даты и времени: year (год), month (месяц), week (неделя), day (день), hour (час), minute (минута), second (секунда), am (до полудня), pm (после полудня). Если требуются слова во множественном числе (что не обязательно), то следует добавить окончание s, например, days, weeks и т.п.
- ❒ Слова: ago (тому назад), now (сейчас), last (последний), next (следующий), this (этот), tomorrow (завтра), yesterday (вчера)
- ❒ Числа и знаки + и –
- ❒ Временные зоны: например, gmt (Greenwich Mean Time — среднее время по Гринвичу), pdt (Pacific Daylight Time — дневное тихоокеанское время)

Далее приведены примеры использования ключевых для формирования значения параметра дата-время функции strtotime():

"now" — текщее время

"now + 48 hours" — через 48 часов от текущего времени; здесь слово now опустить

" + 48 hours" — через 48 часов от текущего времени

"10am + 5 days" — 10 часов до полудня через 5 дней

"3 months ago" — 3 месяца назад

"tomorrow 5pm" — 5 часов по полудни завтра

"last Friday" — предыдущая пятница

"next year gmt" — один год спустя

“2006/4/12” — 12 апреля 2006г. 00:00:00

“2006-4-12” — то же самое

“10 Apr 2006” — то же самое

Предположим, необходимо вывести дату и время, соответствующие 6 часам по полудни 3 месяца и 2 дня спустя. Это можно сделать с помощью следующего кода:

```
<?php
$timestamp="d.m.Y H:i:s";
$t=strtotime("+ 3 months + 2 days 6pm");
```

```
echo date($timestamp, $t);
?>
```

Допустим, требуется узнать, сколько часов между двумя моментами времени, заданными как строковые значения, например, “9 Feb 2006 5am” и “2 Mar 2006 3pm”. Для этого следует перевести строковые значения даты-времени в формат timestamp с помощью функции strtotime(), найти разницу между ними и разделить ее на 3600:

```
<?php
$t1=strtotime("9 Feb 2006 5am");
$t2=strtotime("8 Mar 2006 3pm");
echo ($t2-$t1)/3600;          // 658
?>
```

2.9. Массивы

Массив представляет собой упорядоченный набор данных (элементов), объединяемых под общим именем. Обращение к элементам массива происходит по индексу, который может быть как числовым, так и символьным.

2.9.1. Создание массива

Массив можно создать несколькими способами. Самый простой из них состоит в том, чтобы рядом с именем переменной поставить квадратные скобки и с помощью оператора присваивания задать значение элемента. Например,

```
$myarray[]=”Иван”;
```

В результате будет создан массив с именем $myarray и единственным элементом “Иван”. К этому массиву можно добавить другие элементы различных типов, используя тот же синтаксис:

```
$myarray[]=”Федоров”;
$myarray[]=125.7;
```

В рассматриваемом случае массив $myarray будет содержать уже три элемента — два строковых и один числовой.

При создании массива таким способом PHP автоматически индексирует элементы, используя для этого целые числа, начиная с 0. Индексы (ключи) позволяют получить доступ к значениям элементов массива:

```
$myarray[0];      // ”Иван”
$myarray[1];      // ”Федор”
$myarray[2];      // 125.7
```

При создании массива можно сразу индексировать элементы по своему усмотрению, причем не обязательно с нуля и по порядку:

$myarray[1]=”Иван”;

$myarray[2]=”Федоров”;

$myarray[3]=125.7;

Кроме числовых индексов массивы могут иметь символьные (строковые) индексы, например,

$myarray[‘Имя’]=”Иван”;

$myarray[‘Фимилия’]=”Федоров”;

$myarray[‘Зарплата’]=125.7;

> ***Примечание***
> Числовые индексы могут быть представлены и соответствующими строками. Например, индексы 3 и “3” соответствуют одному и тому же элементу массива, а “03” — другому.

Массив также может быть создан с помощью следующей конструкции:

$*имя_массива*=array([*индекс1*=>] *значение1*[, [*индекс2*=>] *значение2*, …])

Здесь квадратные скобки указывают лишь на то, что заключенный в них элемент не является обязательным.

Примеры:

$myarray=array(“Иван”, “Федоров”, 125.7);

$myarray=array(‘Имя’=>“Иван”, ’Фамилия’=> “Федоров”, ‘Зарплата’=>125.7);

$myarray=array(‘Имя’=>“Иван”, 2=> “Федоров”, 3=>125.7);

В первом случае элементы массива автоматически получат числовые индексы, во втором — указанные символьные индексы, а в третьем — символьные и числовые. При создании массива можно совместно использовать индексы различных типов. Однако каждый элемент имеет только один индекс — числовой или символьный. Если какой-то элемент имеет символьный индекс, то обращаться к этому элементу с помощью его порядкового номера в массиве нельзя.

Для создания регулярных массивов можно использовать функцию диапазона

range (*начало*, *конец*, *шаг*),

которая принимает в качестве параметров:

- *начало* — число или символ начала диапазона
- *конец* — число или символ конца диапазона
- *шаг* — число, соответствующее приращению значений элементов массива (не обязательный параметр, по умолчанию равный 1)

Примеры:

```
range(20, 25);                  // массив из элементов: 20, 21, 22, 23, 24, 25
range(20, 25, 2);               // массив из элементов: 20, , 22, 24
range(25, 22, 2);               // массив из элементов: 25, 23, 21
range(20, 25, 1.5);             // массив из элементов: 20, 21.5, 23, 24.5
range("a", "d");                // массив из элементов: "a", "b", "c" , "d"
range("a", "d", 2);             // массив из элементов: "a",  "c"
range("d", "a", 2);             // массив из элементов: "d",  "b"
range("Борис", "Дмитрий");      // массив из элементов: "Б",  "В", "Г", "Д"
```

Количество элементов (длину) массива можно определить с помощью функции count(*массив*) или sizeof(*массив*).

Пример:

```
$myarray=array("Иван", "Федоров", 125.7);
$n=count($myarray);       // 3
```

Каким бы способом ни был бы создан массив, значения его элементов всегда можно изменить с помощью оператора присваивания.

Чтобы добавить новый элемент к существующему массиву, достаточно использовать следующие выражения:

- $*имя_массива*[]=*значение* — добавляется новый элемент с числовым индексом
- $*имя_массива*[*индекс*]=*значение* — если в массиве нет элемента с указанным индексом, то элемент добавляется, а в противном случае — изменяется (перезаписывается)

Пример:

```
$myarray=array("Имя"=>"Иван",125.7); /* Структура массива:
                                        ["Имя"]=>"Иван"
                                        [0]=>125.7        */

$myarray[]="Профессор";             /*  Структура массива:
                                        ["Имя"]=>"Иван"
                                        [0]=>125.7
                                        [1]=>"Профессор" */

$myarray["тел"]="123-4567";         /*  Структура массива:
                                        ["Имя"]=>"Иван"
```

```
                                    [0]=>125.7
                                    [1]=>”Профессор”
                                    [“тел”]=>”123-4567” */

$myarray[0]="Федоров";              /* Структура массива:
                                    ["Имя"]=>"Иван"
                                    [0]=>”Федоров”
                                    [1]=>”Профессор”
                                    [“тел”]=>”123-4567” */
```

Чтобы удалить элемент из массива или удалить массив целиком, используют функцию unset() с параметром, указывающим на элемент массива или сам массив соответственно.

Пример:

```
$myarray=array(“Один”, “Два”, “Три”);    // массив из трех элементов
unset($myarray[1]);          /* удаление 2-го элемента; $myarray теперь содержит два элемента: “Один” и “Три”  с индексами 0 и 2 соответственно */
unset($myarray);             // переменная $myarray теперь имеет значение null
```

Внимание

При удалении элементов из массива оставшиеся в нем элементы сохраняют свои прежние индексы. Иначе говоря, переиндексация массива (даже только с числовыми индексами) не происходит.

Как уже отмечалось ранее, обращение к элементам массива происходит с помощью индексов. Однако в случае обращения к несуществующему элементу выводится сообщение об ошибке. В ряде случаев это не желательно. Чтобы подавить возможное сообщение об ошибке, используют символ @, например,

```
@$anyelement=$myarray[5];
```

2.9.2. Многомерные массивы

Массивы, рассмотренные выше, являются одномерными. Однако элементы массива могут содержать данные различных типов, в том числе и массивы. Если в качестве элементов некоторого одномерного массива создать массивы, то получится двумерный массив. Обращение к элементам такого массива происходит в соответствии со следующим синтаксисом:

имя_массива[*индекс_уровня1*] [*индекс_уровня2*]

Если массив имеет размерность большую двух, то обращение к массивам имеет аналогичный синтаксис: следует добавить необходимое количество квадратных скобок, заключающих нужные индексы.

Пример:

```
/* Двумерный массив */
$сотрудники=array(
        array("Иван", "Иванов", 100),
        array("Василий", "Васильев", 200),
        array("Федор", "Федоров", 300)
);
echo $сотрудники[2][1];              // Федоров
```

Массив, приведенный в данном примере, можно представить в виде таблицы из трех столбцов и трех строк. Каждой строке соответствует массив значений, который сам является элементом массива $сотрудники. В листинге 2.6 приводится сценарий, выводящий элементы двумерного массива в виде таблицы (рис. 2.13).

Листинг 2.6. Вывод таблицы с элементами двумерного массива (вариант 1)

```
<?php
$сотрудники=array(
array("Иван", "Иванов", 100),
array("Василий", "Васильев", 200),
array("Федор", "Федоров", 300)
);

$str="<table border=1>";
for ($i=0; $i<count($сотрудники); $i++) {
       $str.="<tr>";
       for ($j=0; $j<count($сотрудники[$j]); $j++){
              $str.="<td>".$сотрудники[$i][$j]."</td>";
       }
       $str.="</tr>";
}
$str.="</table>";
echo $str;
?>
```

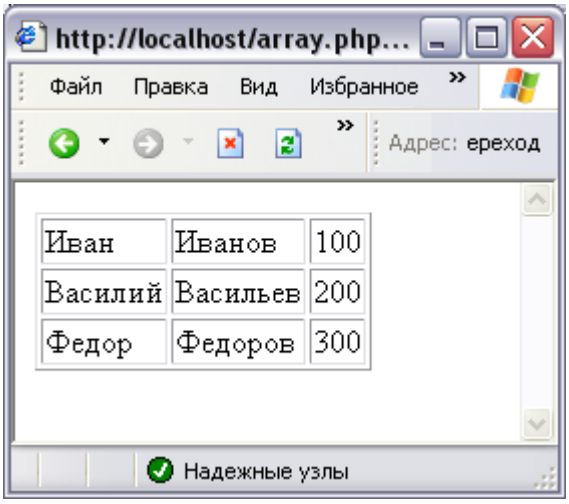

Рис. 2.13. Пример представления двумерного массива в виде таблицы (листинг 2.6)

Другой вариант сценария мы рассмотрим в следующем разделе.

2.9.3. Отображение массивов

Для просмотра структуры и значений элементов массива (это может понадобиться при отладке программ) используются функции print_r() и var_dump(). Эти функции достаточно подробно описывались в разд. 2.3.4.

Разумеется, для вывода значений элементов массива вы можете воспользоваться оператором цикла, например,

```
<?php
$myarray=array("Иван","Федоров", 125.7);
$i=0;
while ($i<count($myarray)) {
        echo $myarray[$i],"<br>";
        $i++;
}
?>
```

Вывод элемента массива с помощью оператора echo производится так же, как и обычной переменной, например,

```
echo $myarray[2];
```

Однако, если требуется вывести элемент массива в составе текстовой строки, то его следует заключить в фигурные скобки.

Пример:

$myarray=array(“Иван”, “Федоров”, 125.7);

echo “Сотрудник {$myarray[1]} получает {$myarray[2]} руб.”; /* выод строки:

“Сотрудник Федоров получает 125.7 руб. “ */

Используя фигурные скобки, код листинга 2.6 можно переписать так, как показано в листинге 2.7.

Листинг 2.7. Вывод таблицы с элементами двумерного массива (вариант 2)

```
<?php
$сотрудники=array(
array("Иван", "Иванов", 100),
array("Василий", "Васильев", 200),
array("Федор", "Федоров", 300)
);

$str="<table border=1>";
for ($i=0; $i<count($сотрудники); $i++) {
        $str.="<tr>";
        for ($j=0; $j<count($сотрудники[$j]); $j++){
                $str.="<td>{$сотрудники[$i][$j]}</td>";
        }
        $str.="</tr>";
}
$str.="</table>";
echo $str;
?>
```

2.9.4. Операции над массивами

Копирование массивов

Чтобы скопировать массив, достаточно просто переменную, содержащую этот массив, присвоить другой переменной. Изменение значения элемента в одном из них никак не отразится на соответствующем элементе в другом массиве.

Пример:

$a=array(5, 2, 4, 3);

$x=$a;

$a[2]=25;

echo $x[2]; // 4

Примечание

В JavaScript присвоение переменной типа массив другой переменной не приводит к копированию массива, а создает лишь новую ссылку (псевдоним) на тот же самый массив.

Сортировка массивов

Первоначально значения элементов массива хранятся в том порядке, как они были созданы. Однако этот порядок можно изменить. Упорядочивать (сортировать) элементы массива можно по индексу и значению. В PHP существует много функций, позволяющих осуществить сортировку массивов различными способами. Здесь мы рассмотрим только некоторые наиболее употребительные из них.

Для сортировки элементов массива с числовыми индексами по возрастанию значений служит функция

sort(*массив* [, *режим*]),

которая возвращает true или false соответственно при удачном и ошибочном завершении операции. Данная функция изменяет указанный массив, а именно изменяет индексы элементов в соответствии с новым порядком их расположения. Если массив имеет символьные индексы, то они заменяются числовыми. Второй необязательный параметр задает режим сортировки (способ сравнения значений элементов) и может принимать следующие значения:

- `SORT_REGULAR` - сравнивать элементы, не изменяя их типы (по умолчанию)
- `SORT_NUMERIC` - сравнивать элементы как числа
- `SORT_STRING` - сравнивать элементы как текстовые строки
- `SORT_LOCALE_STRING` - сравнивать элементы как строки с учетом установленной локали (начиная с версии PHP 5.0.2)

Перечисленные выше константы имеют целочисленные значения 0, 1, 2 и 3 соответственно.

Пример:

```
$myarray=array("Вася", "Анна", 5);      /* Структура массива:
                                        [0]=>"Вася"
                                        [1]=>"Анна"
                                        [2]=> 5 */
sort($myarray);                         /* Структура массива:
                                        [0]=> "Анна"
                                        [1]=> "Вася"
                                        [2]=>5*/
$myarray=array("Вася", "Анна", 5);      /* Исходный массив */
sort($myarray, SORT_STRING );           /* Структура массива:
```

```
                                    [0]=>5
                                    [1]=>"Анна"
                                    [2]=> "Вася"*/
```

При использовании функции sort() исходная связь индекс-значение теряется: элементам в отсортированном массиве назначаются новые индексы в соответствии с их новым положением. Это может оказаться нежелательным, особенно в случае массивов с символьными индексами. Сортировка, при которой эта связь сохраняется, может быть выполнена с помощью функции

asort(*массив* [, *режим*]),

которая по возвращаемой величине и параметрам аналогична функции sort(), но отличается своим действием: элементы массива упорядочиваются по возрастанию значений, но индексы сохраняются прежними.

Пример:

```
$myarray=array("Вася", "Анна", 5);      /* Структура массива:
                                        [0]=>"Вася"
                                        [1]=>"Анна"
                                        [2]=> 5 */
asort($myarray);                        /* Структура массива:
                                        [1]=> "Анна"
                                        [0]=> "Вася"
                                        [2]=>5 */
$myarray=array("Вася", "Анна", 5);      /* Исходный массив */
sort($myarray, SORT_STRING );           /* Структура массива:
                                        [2]=>5
                                        [1]=>"Анна"
                                        [0]=> "Вася"*/
```

Сортировка массива по значениям элементов с сохранением прежних индексов может понадобиться при отображении элементов, но при сохранении прежнего способа обращения к ним.

Рассмотренные выше функции sort() и asort() упорядочивают элементы массива по возрастанию их значений. Для сортировки по убыванию служат соттветственно функции rsort() и arsort().

Сортировать элементы массива можно не только по их значениям, но и по индексам. Как вы уже заметили, порядок расположения элементов в массиве, задаваемый при

его создании или последующей сортировке, не обязательно соответствует порядку индексов элементов. Для сортировки массива по индексам служат функции:

- ksort() — сортирует элементы массива по возрастанию значений индекса
- krsort() — сортирует элементы массива по убыванию значений индекса

Пример:

```
$myarray=array(
        "name"=>"Иван",
        "family"=>"Федоров",
        "bonus"=>125.7
);
ksort($myarray); /* Структура массива:
                    ["bonus"]=>125.7
                    ["family"]=>"Федоров"
                    ["name"]=>"Иван" */
```

Функции ksort() и krsort() имеют такие же параметры и возвращают такие же значения, что и рассмотренные ранее функции сортировки sort() и assort(). Обратите внимание, что применение функций ksort() и krsort() изменяет порядок расположения элементов в массиве, но сохраняет исходную связь индекс-значение.

> ***Внимание***
> Функции сортировки изменяют исходный массив, а не возвращают его отсортированный вариант. Если вам необходимо сохранить исходный массив, то для этого следует предварительно сделать его копию.

Перемещение по массиву

Нередко требуется “пробежаться” по всем элементам массива, чтобы выполнить какие-нибудь операции над ними. Это можно сделать с помощью обычных операторов цикла while, do-while и for (см. разд. 2.5.9). Вместе с тем для этой цели в PHP имеется специальная очень удобная конструкция foreach (для каждого), которая имеет две синтаксические формы:

- foreach (*массив* as $*значение*) {*выражения*};
- foreach (*массив* as $*индекс*=>$*значение*) {*выражения*};

Здесь $*значение* — переменная, выбираемая по своему усмотрению, для представления значения текущего элемента массива; $*индекс* — переменная для представления значения текущего индекса элемента массива. Слово as в выражении, указанном в круглых скобках, является ключевым.

Конструкция foreach последовательно перебирает все элементы указанного ей массива, присваивая их значения переменной $*значение*, а индексы (во втором варианте синтаксиса) — переменной $*индекс*. Эти переменные доступны в коде,

указанном в фигурных скобках. Код, работая с данными переменными, имеет дело с текущим на данной итерации элементом массива.

В листинге 2.8 показано применение foreach в двух своих формах для вывода элементов массива (рис. 2.14).

Листинг 2.8. Применение оператора foreach

```
<?php
$сотрудник=array(
        "Имя"=>"Иван",
        "Фамилия"=>"Иванов",
        "Зарплата"=>10000
);

foreach($сотрудник as $value){
        echo "$value<br>";
}
echo "<p>";
foreach($сотрудник as $key=>$value){
        echo "$key - $value<br>";
}
?>
```

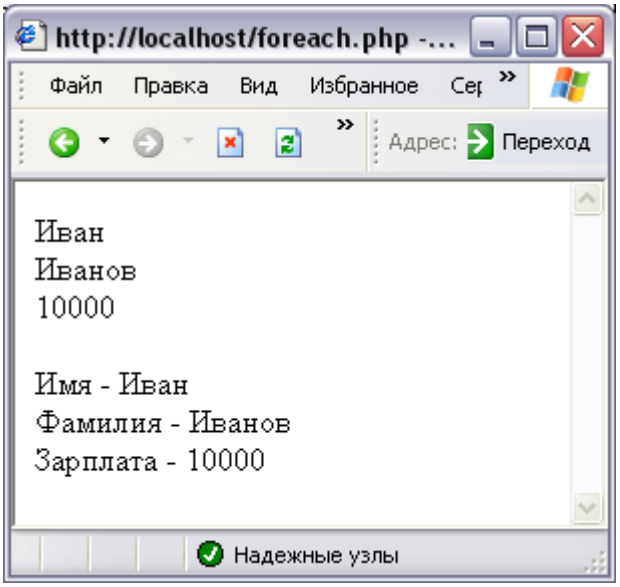

Рис. 2.14. Пример использования оператора foreach

Перебор элементов массива можно выполнить и с помощью специальных функций, которые реализуют перемещение указателя на текущий элемент массива:

- current(*массив*) — возвращает текущее значение элемента массива, не перемещая указатель
- next(*массив*) — перемещает указатель на следующий элемент массива
- prev(*массив*) — перемещает указатель на предыдущий элемент массива
- end(*массив*) — перемещает указатель на последний элемент массива
- reset(*массив*) — перемещает указатель на начальный элемент массива

Если к массиву не применялась функция перемещения указателя, то считается, что он установлен на первый (начальный) элемент.

Пример:

```
$myarray=array("name"=>"Иван","family"=>"Федоров", "bonus"=>125.7);
$x=current($myarray);          // Иван
$x=next($myarray);             // Федоров
$x=reset($myarray);            // Иван
```

Все перечисленные выше функции перемещения указателя возвращают значение элемента массива. Однако применение функций prev() и next() может вывести указатель за пределы массива, и тогда они вернут логическое значение false, а не значение элемента массива. Поэтому эти функции обычно используются совместно с проверкой выхода за границы массива. Например,

```
$x=next($myarray);    // переход к следующему элементу массива
if ($x===false) {           // если выход за конец массива
echo "Выход за конец массива";      // вывод предупреждения
} else {
echo $x;                            // вывод значения элемента массива
}
```

Обратите внимание, что здесь и в других подобных случаях необходимо использовать оператор жесткого равенства, поскольку элемент массива, возвращаемый функцией next(), может, вообще говоря, иметь значения null, "", 0, "0" и "false", но это еще не повод остановить обработку. Оператор жесткого равенства учитывает типы сравниваемых значений (см. разд. 2.6.5).

Запись значений элементов массива в переменные

Чтобы значения элементов массива с числовыми индексами, начинающимися с 0, присвоить переменным из заданного списка, можно использовать следующую конструкцию:

list(*список_переменных*) = *массив*;

При этом создаются переменные, указанные в списке через запятую, и им присваиваются элементы массива с числовыми индексами: первой переменной присваивается значение элемента с индексом 0, второй переменной — значение элемента с индексом 1 и т.д.

Пример:

```
$myarray=array("a", "b", 5, 10);
list($x,$y)=$myarray;
var_dump($x, $y);          // вывод значений и типов переменных $x и $y
```

> ***Примечание***
> Конструкция list() по своему синтаксису похожа на вызов функции, но отличается от него тем, что может быть записана слева от оператора присваивания. Однако выражению с вызовом настоящей функции нельзя присвоить какое бы то ни было значение. Иначе говоря, вызов функции может встречаться только справа от оператора присваивания.

В списке переменных можно использовать элементы массива, например,

```
list($b[0], $b[1], $b[2]) = $a;
```

В этом случае происходит копирование значений элементов массива $a в элементы массива $b. Однако данная операция осуществляется, начиная с крайнего правого элемента массива $a, в результате чего порядок расположения элементов в массиве $b будет противоположен порядку элементов в массиве $a, хотя индексы элементов сохраняются.

Пример:

```
$a=array("a", "b", 5, 10);
list($b[0], $b[1], $b[2])=$a;
var_dump($b);    /* Структура массива $b:
                        [2]=> 5
                        [1]=> "b"
                        [0]=> "a"  */
```

Преобразование массива в текстовую строку

Чтобы значения элементов массива объединить в их строковом представлении в одну текстовую строку, можно использовать функцию

implode(*разделитель*, *массив*)

Параметр *разделитель* является строкой символов, которая вставляется между элементами массива при их объединении в текстовую строку. Если этот параметр не указан, то предполагается пустая строка “”. Данная функция возвращает строку, полученную в результате преобразования.

Пример:

```
$myarray=array("Один", "Два", "Три");
$str=implode($myarray);          // строка “ОдинДваТри”
$str=implode("-", $myarray);     // строка “Один-Два-Три”
```

Преобразование текстовой строки в массив

Для записи фрагментов текстовой строки в элементы массива служит функция

explode(*разделитель*, *строка* [, *количество*])

Параметр *разделитель* является строкой символов, которая рассматривается как разделитель фрагментов строки. Необязательный параметр (о чем говорят квадратные скобки) *количество* указывает максимальное количество элементов в возвращаемом массиве (при этом последний элемент будет содержать весь остаток строки).

Если *разделитель* не содержится в строке, то функция возвращает массив, содержащий единственный элемент, значением которого является исходная строка. Если *разделитель* является пустой строкой, то функция возвращает логическое значение false.

Пример:

```
$str="Один Два Три";
$myarray=explode(" ",$str);      /* Структура массива:
                                 [0]=>”Один”
                                 [1]=> "Два"
                                 [2]=> "Три"  */

$myarray=explode("Два",$str);    /* Структура массива:
                                 [0]=>”Один ”
                                 [1]=> " Три"  */
```

Другие операции над массивами

Здесь я перечислю еще несколько полезных функций для работы с массивами и приведу несколько примеров. Подробное их описание можно найти в справочном руководстве по PHP.

- array_unique(*массив*) — возвращает массив, полученный из указанного в качестве параметра путем удаления из него повторяющихся элементов

  Пример:

  ```
  $myarray=array("a", "b", "a", "c", 5, 5);
  $x=array_unique($myarray);     /* Структура массива:
                                 [0]=>"a"
  ```

[1]=> "b"

[3]=> "c"

[4]=>5 */

- array_sum(*массив*) — возвращает сумму значений элементов массива, указанного в качестве параметра

  Пример:

  $myarray=array(1, 2, "3");

  $x=array_sum($myarray); // 6

- array_slice(*массив*, n1, n2) — возвращает миссив, являющийся частью исходного массива, указанного в качестве параметра. В результатный массив попадет n2 элементов эсходного массива, начиная с элемента, имеющего номер n1.

  Пример:

  $myarray=array(1, 2, 3, 4, 5, 6);

  $x=array_slice($myarray, 2, 3); /* Структура массива:

  [0]=>3

  [1]=> 4

  [2]=> 5 */

  $x=array_slice($myarray, -2, 3); /* Структура массива:

  [0]=>5

  [1]=> 6 */

- array_merge(*массив1*, *массив2*,…, *массивN*) — возвращает массив, получающийся в результате объединения двух или более массивов, указанных в качестве параметров

  Пример:

  $a=array("a", "b", "c");

  $b=array("a", "x", "y");

  $x=array_merge($a, $b); /* Структура массива:

  [0]=>”a”

  [1]=> “b”

  [2]=> “c”

  [3]=>”a”

  [4]=>”x”

  [5]=>”y” */

- array_diff(*массив1*, *массив2*,…, *массивN*) — возвращает массив, состоящий из тех элементов массива, указанного в качестве первого параметра, которых нет в остальных массивах

  Пример:

  ```
  $a=array("a", "b", "c");
  $b=array("a", "x", "y");
  $x=array_diff($a, $b);          /* Структура массива:
                                  [1]=>”b”
                                  [2]=> “c”  */
  ```

- array_flip(*массив*) — возвращает массив, получающийся из указанного в качестве параметра путем перестановки местами индексов и значений

  Пример:

  ```
  $a=array("a", "b", "c");
  $x=array_flip($a);              /* Структура массива:
                                  [“a”]=>0
                                  [“b”]=>1
                                  [“c”]=>2  */
  $a=array("Планета"=>"Земля", "Радиус"=>6378);
  $x=array_flip($a);              /* Структура массива:
                                  [“Земля”]=>”Планета”
                                  [6378]=>”Радиус”  */
  ```

- extract(*массив*) — создает переменные из символьных индексов массива и присваивает им значения соответствующих элементов массива; возвращает количество созданных переменных

  ```
  $a=array("Планета"=>"Земля", "Радиус"=>6378);
  $x=extract($a);
  echo "Наша планета называется $Планета. Ее радиус равен $Радиус км";
  ```

- compact(*список_имен_переменных*) — возвращает массив, созданный из указанных переменных; действие этой функции противоположно действию extract()

  Пример

  ```
  $name="Иван";
  $lastname="Федоров";
  $bonus=1234;
  $x=compact("name", "lastname","bonus");          /* Структура массива:
                                                   ["name"]=>"Иван"
  ```

["lastname"]=>"Федоров"
["bonus"]=> 1234 */

2.10. Глобальные предопределенные переменные

В PHP имеются предопределенные переменные типа array (массив) с глобальной областью видимости. Их еще называют автоглобальными или суперглобальными массивами, поскольку они доступны из любого сценария на языке PHP. Ниже приведен их список.

- $GLOBALS — содержит все глобальные переменные, как предопределенные, так и созданные программистом. Имена глобальных переменных в массиве $GLOBALS являются его символьными индексами (ключами). Поэтому, чтобы получить доступ к глобальной переменной, например, $myvar, достаточно использовть выражение $GLOBALS[“myvar”]. Обратите внимание, что имя переменной в качестве индеса пишется без символа $.
- $_GET — содержит данные, переданные в сценарий на PHP как часть URL-адреса. Данные HTML-форм, переданные методом GET, также сохраняются в массиве $_GET.
- $_POST — содержит данные, переданные в сценарий на PHP из HTML-форм методом POST.
- $_COOKIE — содержит данные, переданные в текущий сценарий через механизм cookie
- $_ENV — содержит переменные окружения, такие как название операционной системы, системный диск и др.; содержимое этого массива зависит от операционной системы
- $_FILES — содержит имена файлов, загруженных методом POST с помощью браузера
- $_SERVER — содержит переменные, установленные Web-сервером, либо непосредственно связанные с окружением выполнения текущего сценария. Эта информация зависит от того, какой Web-сервер используется. Более того, если PHP-сценарий запущен из командной строки, то некоторые переменные заведомо будут недоступны. Таким образом, состав переменных в данном массиве может быть различным. Например, переменная DOCUMENT_ROOT содержит полное имя каталога, в котором Web-сервер ищет запрашиваемые Web-страницы; PHP_SELF — имя файла текущего сценария; REQUEST_METHOD — название метода, который использовался для доступа к странице: “GET”, “POST”, “HEAD”, “PUT”.
- $_SESSION — содержит все переменные сеанса, доступные в текущем сценарии.

- $_REQUEST — содержит все переменные, находящиеся в массивах $_GET, $_POST и $_COOKIE.

Для просмотра содержимого глобальных массивов можно воспользоваться функцией var_dump(), например, var_dump($_SERVER) или конструкцией foreach. Например, для просмотра массива $_SERVER сценарий может иметь такой вид:

```
foreach($_SERVER as $var=>$value){
echo "$var = $value <br>";
}
```

Если, например, серверному сценарию из HTML-формы было передано методом POST содержимое элемента с атрибутом name=”userdata”, то оно может быть получено в сценарии как $_POST[‘userdata’].

Подробное описание содержимого глобальных массивов можно найти в справочном руководстве по PHP. Применение некоторых из них будет рассмотрено в следующей главе.

> ***Внимание***
> По умолчанию глобальные массивы доступны, поскольку в конфигурационном файле php.ini имеется установка register_globals = off. Если заменить ее на register_globals = on, то глобальные массивы будут недоступны.

> ***Примечание***
> В более ранних версиях PHP использовались глобальные массивы с длинными именами: $HTTP_GET_VARS, $HTTP_POST_VARS, $HTTP_COOKIE_VARS, $HTTP_ENV_VARS, $HTTP_FILES_VARS, $HTTP_SERVER_VARS, $HTTP_SESSION_VARS. В PHP 5 они также доступны, хотя и не рекомендуются к использованию. Вместе с тем установка register_long_arrays = off в конфигурационном файле php.ini позволяет отключить их. Если же вы используете старые сценарии, то оставьте установку register_long_arrays = off.

2.11. Функции

Функция это конструкция, которая позволяет оформить блок программного кода для многократного использования. Такой блок кода (тело функции) можно озаглавить путем назначения ему имени и указать при необходимости параметры (значения переменных и выражений), с которыми он будет работать. Тело функции можно построить так, чтобы она возвращала во внешнюю программу, из которой она была вызвана, некоторое значение. Впрочем, функция в PHP (как и в JavaScript) может и не требовать параметров, а также ничего не возвращать. Выражение вызова функции (имя функции с возможным указанием передаваемых ей параметров) можно использовать в простых и сложных выражениях языка PHP, если эта функция что-нибудь возвращает.

Культура программирования конкретных приложений настоятельно рекомендует создавать функции, которые могут использоваться в различных приложениях (принцип: делайте как можно более абстрактные функции). С другой стороны, для

достижения наибольшей эффективности кода приложения старайтесь создавать функции, выполняющие специфические конкретные действия (принцип: действия функции должны быть как можно конкретнее). Искусство программирования выражается в компромиссных решениях, сочетающих, так или иначе, эти два противоположных принципа.

2.11.1. Пользовательские функции

Пользовательские функции — это функции, которые определяются программистом. Создать функцию означает написать в программном коде ее определение: имя, список параметров (не обязательно) и программный код (тело функции), который будет выполнен при вызове этой функции. Итак, с программной точки зрения с функцией связано две вещи: определение и вызов. Рассматривая встроенные функции, мы всегда имели дело с выражениями вызова этих функций. Их определения были скрыты от нас, поскольку созданы не нами.

Определение (описание) функции начинается ключевым словом `function`. Точнее, описание функции имеет следующий синтаксис:

```
function имя_функции(параметры){
  код
}
```

Имя функции выбирается так же, как и имя переменной, за исключением того, что недопустимо первым символом в имени функции использовать символ $. Не допустимо также использовать в качестве имени ключевые слова языка PHP. За именем функции обязательно стоит пара круглых скобок. Программный код (тело) функции заключается в фигурные скобки. Они определяют группу выражений, которые относятся к коду именно этой функции. Если функция принимает параметры, то список их имен (идентификаторов) указывается в круглых скобках около имени функции. Имена параметров выбираются согласно тем же требованиям, что и имена обычных переменных. Если параметров несколько, то в списке они разделяются запятыми. Если параметры для данной функции не предусмотрены, то в круглых скобках около имени функции ничего не пишут.

Если требуется, чтобы функция возвращала некоторое значение, то в ее теле (в программном коде, заключенном в фигурные скобки) используется оператор возврата `return` с указанием справа от него того, что следует возвратить. В качестве возвращаемой величины может выступать любое выражение: простое значение, имя переменной или вычисляемое выражение. Оператор `return` может встречаться в коде функции несколько раз. Если оператор `return` не использовать, или не указывать справа от него возвращаемую величину, то функция ничего не будет возвращать.

Пример:

```
function S_rectangle($width, $height) {
        $S=$width* $height;
        return $S;
```

}

Здесь функция с именем S_rectangle принимает два параметра — $width и $height, вычисляет и возвращает произведение их значений .

Определение функции, будучи помещенным в программу, усваивается интерпретатором, но сама функция (ее код или тело) не выполняется. Чтобы выполнить функцию, определение которой задано, необходимо написать в программе выражение вызова этой функции. Оно имеет следующий синтаксис:

имя_функции(*параметры*);

Пример:

```
S_rectangle(3, 5);          // возвращаемое значение равно 15
```

Имя функции в ее вызове должно совпадать с именем ранее определенной функции. Однако совпадение с точностью до регистра не обязательно. Например, выражения S_rectange(3, 5) и s_Rectangle(3, 5) эквивалентны.

> ***Примечание***
> В JavaScript имя функции в ее вызове и в определении должны сровпадать с точностью до регистра

Параметры, если они заданы в определении функции, в вызове этой функции представляются конкретными значениями, переменными или выражениями.

Пример:

```
$x=5;
$S=S_rectangle(3, $x+2);
```

Параметры функции могут быть переданы по значению и по ссылке. По умолчанию они передаются по значению. Это означает, что изменение в теле функции значения переменной, переданной в качестве параметра, никак не отразится на значении этой же переменной во внешнем коде. Кроме того, любая переменная, созданная в теле функции, является локальной, т.е. видимой только в пределах ее кода.

Пример:

```
<?php
$x=5;
function myfunc($arg){
        $arg=$arg+2;
        $y=10;
        return    $arg;
}
myfunc($x);       // возвращает 7
echo $x;          // выводит 5 (прежнее значение)
echo $y;          /* ничего не выводится, т.к.
```

```
                значение $y вне функции myfunc() равно null
?>
```

При передаче параметров по значению в функции создаются локальные переменные с такими же именами, что и имена формальных параметров, указанных в определении функции. Это — локальные переменные, отличные от внешних, даже если их имена совпадают. Следующий код эквивалентен приведенному выше.

```
<?php
$x=5;
function myfunc($x){
        $x=$x+2;
        $y=10;
        return  $x;
}
myfunc($x);     // возвращает 7
echo $x;        // выводит 5 (прежнее значение)
echo $y;        /* ничего не выводится, т.к.
                 значение $y вне функции myfunc() равно null
?>
```

Чтобы изменения параметров, произведенные в теле функции, были видны и за пределами ее тела, необходимо передавать их по ссылке. Для указания того, что параметр передается по ссылке, перед его именем в списке параметров в определении функции следует написать символ & (амперсанд):

function *имя_функции*(&$arg1, &$arg2, …){

  КОД

}

Пример:

```
<?php
$x=5;
function myfunc(&$arg){
        $arg=$arg+2;
        return  $arg;
}
myfunc($x);
echo $x;        // 7 (новое значение)
```

```
?>
```

При передаче параметров по ссылке формальному параметру, указанному в определении функции, передается не значение, а адрес внешней переменной.

> ***Примечание***
> В JavaScript параметры передаются функции только по значению.

При передаче функции большего количества параметров, чем их предусмотрено в определении, лишние параметры будут просто проигнорированы. Если передается меньшее количество, то пропущенным параметрам присваивается значение null и при этом может быть выведено соответствующее предупреждающее сообщение, что, однако, не приведет к остановке выполнения кода. Вместе с тем, в определении функции можно указать значения параметров, принимаемые по умолчанию. Это полезно на тот случай, если вы укажете в вызове функции меньшее количество параметров, чем их предусмотрено в определении. Для задания значений по умолчанию используется оператор присваивания непосредственно в списке формальных параметров определения функции.

Пример:

```
<?php
function S_rectangle($width=10, $height=5) {
        $S=$width* $height;
        return $S;
}
echo S_rectangle();          // 50
echo “<br>”;
echo Srectangle(3);          // 15
?>
```

> ***Внимание***
> 1. При задании для параметров в определении функций значений по умолчанию, последние должны быть простыми значениями, а не переменными, вычисляемыми выражениями или вызовами функций и методов объектов.
>
> 2. Если в списке используются параметры без значений по умолчанию, а также параметры с заданными значениями по умолчанию, то последние должны располагаться в правой части списка, т.е. за теми параметрами, для которых значения по умолчанию не предусмотрены.

При создании функций, способных работать с переменным по длине списком параметров, можно применять следующие встроенные функции:

- func_num_args() — возвращает количество параметров, переданных функции, из которой она была вызвана

  Пример:

```
<?php
function S_rectangle($width, $height) {
        if (func_num_args()==0){// если нет параметров
                $width=10;
                $height=10;
        }
        if (func_num_args()==1){// если один параметр
                $height=10;
        }
        return $width* $height;
}
echo S_rectangle();      // 100
echo “<br>”;
echo S_rectangle(2);    // 20

?>
```

- func_get_args() — возвращает массив значений параметров, переданных функции, из которой она была вызвана.

```
<?php
function S_rectangle($width, $height) {
/* значения параметров по умолчанию: *./
        $width=10;
        $height=10;
        $args=func_get_args();                  // массив значений параметров
        $narg=count(func_get_args());       // длина массива значений параметров
        if ($narg==1) $width=$args[0];      // если один параметр
        if ($narg==2) {$width=$args[0]; $height=$args[1];} // если два параметра
        return $width* $height;
}
echo S_rectangle();      // 100
echo "<br>";
echo S_rectangle(2);    // 20
echo "<br>";
echo S_rectangle(2, 3);// 6
```

```
?>
```

- func_get_arg(*номер_параметра*) — возвращает значение параметра, указанного с помощью его номера. Если *номер_параметра* превышает длину списка переданных параметров, то данная функция возвращает false. Обычно эта функция используется совместно с func_get_args() и func_num_args().

Пример:

```
<?php
function S_rectangle($width, $height) {
        echo "width=".func_get_arg(0)." height=".func_get_arg(1);
}
S_rectangle();          // width= height=
echo "<br>";
S_rectangle(2);         // width=2 height=
echo "<br>";
S_rectangle(2, 3);      // width=2 height=3
?>
```

Внимание

При вызове функции с некорректным количеством параметров (даже если вы обрабатываете эти ситуации в теле функции) может выводиться предупреждающее сообщение “**Warning**: Missing argument…”. Однако программа при этом продолжает выполняться. Если вы желаете предотвратить вывод подобных сообщений, то перед вызовом функции следует указать символ @, например, @S_rectangle().

2.11.2. Переменные функции

Если рядом с именем переменной написать круглые скобки, то интерпретатор PHP возьмет значение этой переменной и попытается выполнить одноименную функцию. Иначе говоря, если строковое значение переменной совпадает с именем определенной функции и рядом с переменной указаны круглые скобки, то будет совершена попытка выполнить эту функцию. В круглых скобках при необходимости можно указать список параметров.

Пример:

```
<?php
function S_rectangle($width, $height) {  // определение функции
        return $width*$height;
}
$x="S_rectangle";       // значение переменной $x совпадает с именем функции
echo $x(2, 3);          // 6
?>
```

Имена функций можно хранить и в массивах. Например,

```
<?php
$afuncname=array("sqrt","sin","cos");        // массив имен функций
$x=0.5;
$i=0;
while ($i<count($afuncname)){
        echo $afuncname[$i]($x);
        echo "<br>";
        $i++;
}
?>
```

Данный код выводит на экран значения математических функций sqrt(0.5), sin(0.5) и cos(0.5), имена которых заданы в массиве $afuncname.

2.11.3. Встроенные функции

В PHP есть большое множество *встроенных функций*. Некоторые из них уже были рассмотрены в предыдущих разделах. Подробное описание других функций можно найти в справочном руководстве по PHP (см. разд. 1.7.4). Следует заметить, что некоторые из встроенных функций доступны при обычной установке PHP, другие требуют особых, хотя и несложных, настроек модуля PHP. Все рассмотренные ранее встроенные функции PHP доступны всегда без специфических настроек модуля PHP. Если обратиться к справочной информации по функциям PHP, то там можно найти сведения о их доступности. Для доступа к некоторым функциям, например, работы с базами данных, графической информацией и др. требуется подключение специальных библиотек (см. разд. 1.2.3) .

2.11.4. Есть ли такая функция?

Вызов функции, встроенной или пользовательской, возможен, если имеется ее определение. Иногда в сценарии требуется узнать, имеется ли определение той или иной функции. Это можно сделать с помощью встроенной функции function_exists(*имя_функции*) . Данная функция возвращает true, если определение указанной в параметре функции существует, и false — в противном случае. Однако следует иметь в виду, что function_exists(*имя_функции*) возвращает true даже для существующих функций, но которые нельзя использовать из-за настроек конфигурации.

2.12. Классы и объекты

Язык PHP 5 является объекто-ориентированным языком программирования. Объектно-ориентированное программирование (ООП) предполагает особый

синтаксис и подходы к разработке и анализу программ. Основными элементами программ в ООП являются объекты. Обычно они, так или иначе, соответствуют предметам действительности, моделируемой посредством программ. Например, автомобиль, банковский счет, форму для ввода данных и персонажи компьютерной игры можно рассматривать как объекты. С формальной же точки зрения объект является неким контейнером, в котором сгруппированы свойства и методы объекта. Свойства и методы можно понимать как переменные и функции, связанные с объектом или, если хотите, присущие ему.

Наряду с понятием объекта в ООП используется понятие класса. Класс это описание, своего рода шаблон, по которому создаются объекты. Иногда вместо термина “класс” применяют термин “объект”, а вместо термина “объект” — “экземпляр объекта”. В этой главе мы будем говорить о классах и объектах.

С точки зрения систем программирования класс является обобщением таких понятий, как функция и структура данных. И то, и другое объединяются понятием и соответствущей синтаксической конструкцией, которые называются классом.

Между классами может быть установлено отношение наследования. Мы можем создать класс A как контейнер, содержащий описание свойств некоторого моделируемого предмета. Эти свойства могут быть различного уровня. Так, они могут восприниматься нашими органами чувств более или менее непосредственно, обнаруживаться с помощью датчиков и измерительных приборов, а также “вычисляться” умом посредством усвоенных концепций и теорий. Класс A может быть более или менее абстрактным (общим). Возможно, мы захотим уточнить его, сохранив некоторые исходные свойства и добавив новые. Речь идет о создании нового класса B, связанного каким-то образом с исходным. Если класс B является наследником (потомком) класса A, то он имеет (наследует) все его свойства и методы, а также, воможно, обладает и собственными свойствами и методами. При этом класс A называют родительским или базовым для класса B. Например, можно создать два класса автомобилей: Грузовик и Легковой. Ни один из них не является наследником другого. Поскольку между ними много общего, можно сначала создать общий класс Автомобиль, а затем два класса-наследника — Грузовик и Легковой.

> ***Примечание***
> PHP 5 не вполной мере реализует принципы ООП в отличие от, например, классического ООП-языка C++. Так, PHP не поддерживает полиморфизм (в одном классе не может быть двух и более одноименных методов с различными наборами параметров), а также множественное наследование (каждый класс не может иметь более одного родительского класса). Тем не менее, ООП-возможности PHP 5 существенно шире, чем в JavaScript.

2.12.1. Определение класса

Определение класса, который не является наследником другого класса, имеет следующий синтаксис:

```
class имя_класса {
       определение свойств
```

```
        определение методов
}
```

Чтобы определить класс-наследник, используется такой синтаксис:

```
class имя_класса-наследника extends имя_класса-родителя {
        определение свойств
        определение методов
}
```

В объекте, созданном на основе класса *имя_класса-наследника,* доступны все свойства и методы класса *имя_класса-родителя.* Однако обратное утверждение не верно: в объекте родительского класса свойства и методы его наследников не доступны.

Свойства и методы

Свойства в определении класса задаются как обычные переменные. При этом вы можете использовать оператор присваивания, чтобы установить для свойств значения по умолчанию. Значения должны быть простыми (различных типов), а не задаваться посредством вычисляемых выражений. Вместе с тем, можно и рекомендуется использовать ключевое слово var для объявления переменных. Вот пример определения класса с несколькими свойствами-переменными:

```
class myobject{
        var $x;
        var $y=5;
        var $color="красный"
        var $address("Санкт-Птербург", "Литейный пр. ", 22, 3);
}
```

> ***Внимание***
>
> Спецификатор (ключевое слово) var перед именем свойства означает доступность этого свойства из-за пределов кода класса. Вместо var можно использовать спецификатор public (публичный, общедоступный). Допустимы также и другие спецификаторы: private и protected, но они задают ограничение доступа к свойствам (см. разд. 2.13.3). При объявлении свойств указание спецификатора обязательно.

Методы объекта также являются его свойствами, но задаются не как переменные, а как функции. Методы обычно используются, чтобы изменить значения свойств объекта и/или произвести какие-нибудь другие действия. Если в методе (функции) требуется обратиться к какому-нибудь свойству (переменной) этого же обекта, то перед именем этого свойства следует указать префикс $this->. Например, выражение $this->color означает обращение к свойству $color этого же класса (this — этот). Метод задается в определении класса как обычная функция с использованием ключевого слова function. В следующем примере описывается простой класс для создания счета.

```
class myaccount {
	var $summa=0;			// остаток на счете
	function addsum($sum) {		// добавление к сумме на счете
	$this->summa = $this->summa + $sum;
	echo "Добавлено $sum рублей";
}
}
```

Данный класс имеет одно свойство $summa и один метод addsum(). Он предназначен для хранения некоторого числа, понимаемого как остаток денежной суммы на некотором счете, т.е. как состояние счета. С помощью метода addsum() можно изменить текущее состояние счета, указав в качестве параметра число, которое следует прибавить к остатку на счете (если это число положительное), или вычесть из остатка (если число отрицательное).

Конструктор

В определении класса при необходимости можно задать один особый метод, называемый конструктором. Конструктор выполняется при создании объекта на основе данного класса, хотя объект можно создать и не определяя конструктор явно. Обычно конструктор определяют явным образом, если при создании объекта требуется сразу же выполнить некоторый код, установить начальные значения свойств объекта и/или передать значения его свойств как параметры. Формально конструктор является функцией, имя которой не произвольно, а является ключевым словом __constuct (с двумя ведущими символами подчеркивания) или же должно совпадать с именем класса.

Пример:

```
class account {
	var $summa=0;			// свойство
	function __construct(){		// конструктор
		$x=1000;
		$this->summa=$x;
		echo "На вашем счете $x руб.";
	}
	function addsum($sum) {		// метод: добавление к сумме на счете
		$this->summa = $this->summa + $sum;
		echo "Добавлено $sum рублей";
	}
}
```

Здесь конструктор устанавливает текущее значение счета $summa (хотя значение по умолчанию для этого свойства уже установлено) и выводит соответствующее сообщение. Заметим, что в данном примере в качестве имени конструктора можно было бы выбрать account, т.е. имя, совпадающее с именем класса.

Как уже отмечалось, конструктор в явном виде используется еще и для того, чтобы передать создаваемому объекту значения его свойств подобно тому, как это делается в случае обычных функций. Вот пример:

```
class account {
       var $summa;                          // свойство
       function __construct($summa){       // конструктор
              $this->summa=$summa;
              echo "На вашем счете $this->summa руб.";
       }
       function addsum($sum) {      // метод: добавление к сумме на счете
              $this->summa = $this->summa + $sum;
              echo "Добавлено $sum рублей";
       }
}
```

Здесь в определении класса объявлено свойство $summa. Конструктор выполняет две вещи:

- обеспечивает передачу создаваемому объекту значения свойства $summa как параметра
- вывод сообщения о состоянии счета на момент создания объекта

Теперь рассмотрим создание двух классов — родительского с именем account и его наследника с именем myaccount (листинг 2.5).

Листинг 2.9. Пример родительского класса и его наследника

```
<?php
class account {
       var $summa;                  // объявление свойства
       function addsum($sum) {      // метод: добавление к сумме на счете
              $this->summa = $this->summa + $sum;
       }
}

class myaccount extends account{
       var $name;    // объявление свойства
       function __construct($nam, $sum){   //  конструктор
              $this->name=$nam;
              $this->summa=$sum;
       }
       function set_name($name) {   /* метод: установка имени владедьца
                                       счета */
              $this->name = $name;
```

```
        }
}
?>
```

Класс account содержит только свойство $summa для хранения суммы счета и метод addsum() для его пополнения. Класс myaccount уточняет (дополняет) родительский класс account свойством $name для хранения имени владельца счета и методом set_name() для его установки. Конструктор класса myaccount обеспечивает установку имени и суммы на счете при создании объекта myaccount. Создание и применение объектов будет рассмотрено в следующем разделе.

Разумеется, рассмотренная задача могла быть решена и на основе одного, а не двух классов. Скорее всего так и следовало бы поступить, но рассмотренный выше код призван лишь проиллюстрировать наследование свойств классов.

2.12.2. Применение объектов

Если в сценарии находится определение класса, то его можно использовать для создания объектов. Определение класса может располагаться в сценарии непосредственно, либо находиться в другом файле, содержимое которого включается в текущий сценарий с помощью функции include().

Объект создается на основе определенного класса с помощью оператора new:

$имя_переменной = new *имя_класса*(*параметры*);

Оператор присваивания здесь используется, чтобы присвоить переменной ссылку на созданный объект, которую затем удобно применять для доступа к его свойствам и методам. Эта переменная имеет тип object. Для доступа к свойствам и методам объекта применяется так называемый стрелочный (а не точечный, как в JavaScript) синтаксис:

объект->свойство;

объект->метод();

Пример:

```
<?php
class account {
… // код из листинга 2.9
}
class myaccount extends account{
... // код из листинга 2.9
}
$w=new myaccount("Вася",1000);          // Положили Васе на счет 1000
$w->addsum(500);                        // Добавили 500
$w->set_name("Петя");                   // Заменили имя на Петя
```

```
echo "Имя: $w->name".". Сумма: $w->summa рублей";       /* Вывод строки:
                                                    "Имя: Петя. Сумма: 1500 рублей"  */
?>
```

В описании класса myaccount (см. листинг 2.9) конструктор организует прием значений двух параметров — имя и начальное значение счета. Например, в результате выполнения выражения

```
$w=new myaccount("Вася",1000);
```

свойствам $name и $summa будет присвоены значения “Вася” и 1000 сразу же при создании объекта. Кроме того, переменной $w присвоена ссылка на созданный объект, которая была использована в выражениях вызова методов:

```
$w->addSum(500);
$w->set_name("Петя");
```

Обратите внимание, что первый метод определен в родительском классе, а второй — в его классе-наследнике. Тем не менее мы можем вызавать их как методы объекта класса-наследника myaccount.

2.12.3. Ограничение доступа к свойствам и методам

Рассмотренные выше свойства и методы класса доступны из сценария как свойства и методы созданного на основе этого класса объекта. Иначе говоря, при синтаксически правильном обращении свойства и методы объекта доступны из программного кода, являющегося внешним по отношению к коду описания класса. Однако в ряде случаев требуется, чтобы некоторые свойства и/или методы объекта были недоступны из внешнего кода. Так например, считается, что прямой доступ к свойствам-переменным класаса — плохой стиль программирования. Лучше обеспечивать доступ к свойствам извне с помощью специальных методов класса.

В PHP можно установить ограничение доступа к свойствам (переменным и функциям) класса с помощью следующих спецификаторов (ключевых слов):

- private (закрытый, частный) — запрещает доступ к свойствам и методам класса за его пределами (из сценариев и других классов).
- protected (защищенный) — запрещает доступ к свойствам и методам класса за его пределами, за исключением его классов-наследников.

Примеры:

```
private $x;
protected $myvar;
protected function myfunc($arg){…}
```

Перепишем определение класса account (см. листинг 2.9) так, чтобы нельзя было воспользоваться его методом непосредственно:

```
class account {
```

```
        var $summa;             // объявление свойства
        protected function addsum($sum){ /* защищенный метод:
                                        добавление к сумме на счете */
                $this->summa = $this->summa + $sum;
        }
}
```

Теперь попытка выполнить, например, код

```
$x=new account();
$x->addsum(500);
```

приведет к выводу сообщения об ошибке. Однако изменить значение свойства $summa можно из кода класса-наследника myaccount, если добавить к нему соответствующий метод:

```
class myaccount extends account{
        var $name;          // объявление свойства
        function __construct($nam, $sum){                   //  конструктор
                $this->name=$nam;
                $this->summa=$sum;
        }
        function set_name($name) {          // метод: добавление к сумме на счете
                $this->name = $name;
        }
        function add_sum($sum) {            // метод: добавление к сумме на счете
                $this->addsum($sum);        // вызов защищенного метода
        }
}
```

Здесь метод add_sum($sum) просто вызывает защищенный метод addsum($sum) родительского класса account.

> ***Внимание***
> Спецификатор var означает, что следующее за ним свойство не имеет ограничений доступа. Вместо него можно использовать эквивалентный спецификатор public.

2.12.4. Клонирование и удаление объектов

Для клонирования (копирования) объекта используется оператор

$*объект_копия* = clone $*исходный_объект*;

Этот оператор вызывает специальный метод __clone() (два символа подчеркивания), который по умолчанию просто создает новый объект со свойствами исходного.

Пример:

$w->new mycount(“Саша”, 500); // см. листинг 2.5.

$q-> clone $w; // $q — копия объекта $w

$q->add_sum(500);

$q->set_name("Маша");

echo "Имя: $q->name".". Сумма: $q->summa";

Однако метод __clone() можно переопределить подобно тому, как это делается с конструктором (методом __construct()).

Удалить ранее созданный объект можно с помощью функции unset($*объект*).

Пример:

$w->new mycount(“Саша”, 500); // создание объекта (см. листинг 2.5).

unset($w); // удаление объекта $w

Нередко при удалении объекта требуется выполнить какие-то действия, например, записать данные в файл или базу данных, закрыть файлы и т.п. В подобных случаях можно в классе определить код специального метода __destruct() (два символа подчеркивания), который выполнится перед тем, как объект будет удален. Этот метод называют деструктором.

Пример:

```
class account {
       var $summa;             // объявление свойства
       function _destructor(){
       echo "Объект уничтожен";
       }
       protected function addsum($sum){ /* защищенный метод:
                                           добавление к сумме на счете */
              $this->summa = $this->summa + $sum;
       }
}
```

2.12.5. Использование методов несозданных объектов

Чтобы обратиться к методу или свойству класса, объкт для которого еще не создан, можно использовать следующий синтаксис:

имя_класса::*свойство*();

имя_класса::*метод*();

В следующем примере определен класс с единственным методом, выводящим сообщение, и далее вызов этого метода без предварительного создания объекта данного класса.

```
<?php
class myclass {
        function mymethod(){
                echo "Привет из класса myclass";
        }
}
myclass::mymethod();
?>
```

2.12.6. Обработка исключений

Обработка ошибок в программах на языке PHP происходит на основе так называемого механизма исключительных ситуаций, который использует встоенный класс Exception (исключение). Вы можете определить некоторые нежелательные (исключительные) ситуации и обработать их по своему усмотрению. Если этого не делать, то при возникновении ошибок интерпретатор PHP обработает их некоторым стандартным образом с выводом соответствующих сообщений.

Обратимся в качестве примера к определению класса account, приведенному в листинге 2.9. Метод addsum() в этом классе позволяет изменять состояние счета (свойство $summa). Предположим, что состояние счета не должно быть отрицательным. Иначе говоря, отрицательное значение свойства $summa определяется нами как исключительная ситуация, которую следует обработать особым образом. В листинге 2.10 обработка данной ситуации состоит в выводе сообщения и прекращении выполнения сценария.

Листинг 2.10. Пример обработки исключений

```
<?php
class account {
       var $summa;                  // объявление свойства
       function addsum($sum) {      // метод: добавление к сумме на счете
              $this->summa = $this->summa + $sum;
              try {
                     if ($this->summa < 0) { // если исключение
                            throw new Exception ("Вы должник!");
                     }
              } catch (Exception $e) {
                     exit($e->getMessage());
              }
```

```
        }
}
$w=new account();     // создание объекта класса account
$w->addsum(-100);     // снять со счета 100
?>
```

Здесь в коде метода addsum() содержатся блоки:

- try (попытка) — содержит оператор условия if; если результат проверки условия равен true, то генерируется исключение и выполняется выражение, следующее за ключевым словом throw, в данном случае — создание нового объекта Exception с передачей ему текста сообщения.
- catch (перехват) — указывается объект класса Exception, созданный для данного исключения, а также блок кода, который следует выполнить, если исключительная ситуация возникла. В примере это — вызов метода getMessage(), который возвращает сообщение о ситуации, хранящееся во встроенном классе Exception, вывод этого сообщения и прекращение выполнения сценария с помощью функции exit(). Если исключение не сгенерировано, то блок кода coatch не выполняется.

2.12.7. Пример класса формы

Рассмотрим в качестве примера программное создание HTML-формы на основе применения класса, содержащего следующие свойства и методы:

- $afld — массив полей ввода данных
- $action — значение атрибута action тега <FORM>, которое используется для указания серверной программы, обрабатывающей данные формы.
- $submitlabel — надпись на кнопке типа submit для отправки данных на сервер
- $nfld — количество полей в форме
- __construct() — конструктор для присвоения значений свойствам $action и $submitlabel при создании объекта формы
- addfld() — метод, добавляющий поле в форму
- showform() — метод, отображающий форму

Таким образом, создаваемая форма будет содержать кнопку типа submit и поля ввода данных типа input, которые при необходимости можно добавлять к форме. В листинге 2.11 приведено определение класса Form и сценарий, добавляющий в форму два поля ввода данных, а затем отображающий ее в окне браузера (рис. 2.15).

Листинг 2.11. Пример создания HTML-фориы на основе класса

```
<?php
```

```
class Form {
/* Класс формы */
	var $afld=array();
	var $action;
	var $submitlabel="Отправить";
	var $nfld=0;
/* Конструктор */
	function __construct($action, $submitlabel){
		$this->action=$action;
		$this->submitlabel=$submitlabel;
	}
/* Добавление полей */
	function addfld($name, $label){
		$this->afld[$this->nfld]["name"]=$name;
		$this->afld[$this->nfld]["label"]=$label;
		$this->nfld=$nfld + 1;
	}
/* Отображение формы */
	function showform(){
		echo "<FORM action={$this->action} method=POST>";
		for($i=1; $i<=count($this->afld); $i++){
			echo "{$this->afld[$i-1]['label']}: ";
			echo "<input type=text
				name={$this->afld[$i-1]['name']}><br>";
		}
		echo "<input type=submit value='$this->submitlabel'>";
		echo "</FORM>";
	}
}
/* Создание объекта класса Form */
$f=new Form("http://localhost/myform.php", 'Отправить данные');
/* Добавление полей */
$f->addfld("username","Имя");
$f->addfld("address","Адрес");
/* Отображение формы */
$f->showform();
?>
```

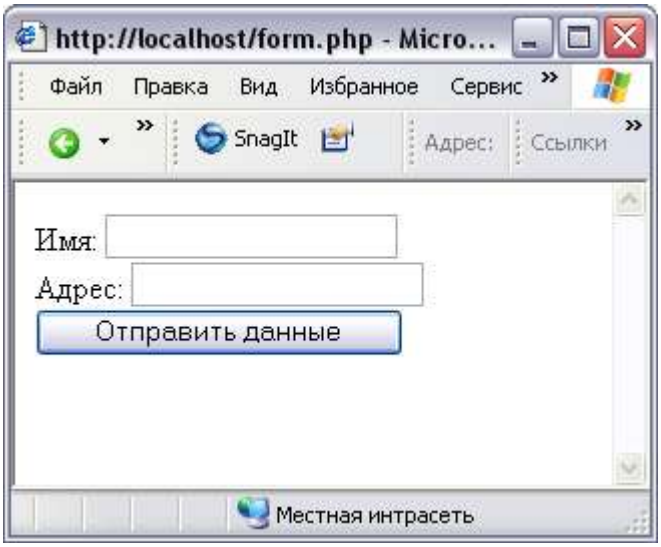

Рис. 2.15. Форма, сгенерированная сценарием листинга 2.11

2.13. Выполнение PHP-кода в текстовых строках

Можно написать или сгенерировать с помощью сценария текстовую строку, содержащую PHP-код, а затем, когда это потребуется, выполнить его. Например, такие строки могут храниться в базе данных. В этом случае их можно считать в переменную для последующего выполнения.

Для выполнения PHP-кода в текстовой строке служит функция eval(*строка*).

Пример:

```
<?php
$x=2; $y=3;
$str = '$z = $x + $y;';
eval($str);    // вычисляет выражение $z = $x + $y;
echo $z;       // выводит число 5
 ?>
```

Обратите внимание, что значение переменной $str заключено в одинарные кавычки, чтобы заключенные в них переменные не интерпретировались. Функция eval($str) вычисляет выражение, являющееся значением переменной $str.

Функция eval(*строка*) может возвращать значение, указанное в строке справа от оператора return, если, разумеется, он используется. В противном случае возвращается null (если не было ошибок) или false (при возникновении ошибки). Если возникает фатальная ошибка, происходит прекращение работы всего сценария.

Пример:

```
<?php
$x=2; $y=3;
$str = '$z = $x + $y; return $z;';
echo eval($str);// выводится возвращаемое функцией eval() значение $z
?>
```

Обратите внимание, что значение переменной $str заключено в одинарные кавычки. При использовании двойных кавычек значение $str было бы равно " =2 + 3; return ;", а при выполнении функции eval($str) появилось бы сообщение об ошибке: **Parse error**: parse error, unexpected '=' in **c:\Inetpub\wwwroot\eval.php(4) : eval()'d code** on line **1** (Ошибка преобразования: недопустимый символ '=' в c:\Inetpub\wwwroot\eval.php(4): код строки 1 функции eval()).

Из приведенных выше примеров видно, что с помощью функции eval() можно создавать новые переменные и присваивать им значения. Вот еще один пример:

```
<?php
$x="newvar";
eval('$'.$x.'=123;');
echo $newvar; // выводится число 123
?>
```

Примечание

Аналогичная функция eval() имеется и в JavaScript.