Глава 5

Работа с данными

В этой главе...

- Что такое типы данных
- Выполнение арифметических операций
- Обработка текстовых строк
- Использование даты и времени

Переменные могут содержать значения различных типов. В свою очередь, над данными различных типов можно выполнять различные операции. Например, можно складывать два числа — `1+2`. Однако сложение двух символов (`a+b`) выполняется совсем по-другому. В этой главе вы узнаете, какие типы данных существуют в языке PHP и как их можно использовать.

Типы данных

В переменных PHP можно хранить данные следующих простых типов.

- ✓ **Целый тип (integer)** позволяет оперировать с целыми числами (без дробной части), такими как `-43`, `0`, `1`, `27` или `5438`. Допустимый диапазон целочисленных значений в общем случае зависит от операционной системы. Однако обычно допустимые значения переменных целочисленного типа лежат в пределе от –2 до 2 миллиардов.
- ✓ **Тип с плавающей точкой (floating point)** позволяет манипулировать числами с дробной частью, например `5.24` или `123.456789`. Такие числа часто называются *действительными* (real number) или *числами с плавающей точкой* (float).
- ✓ **Строковый тип (string)** обеспечивает хранение последовательности символов, например `привет`. На длину текстовой строки практически не накладывается каких-либо ограничений.
- ✓ К **булевому (логическому) типу (boolean)** относятся два значения: TRUE (истина) и FALSE (ложь). Данный тип более подробно рассматривается ниже.

Присваивание типов данных

В большинстве языков программирования требуется, чтобы перед использованием каждая переменная была предварительно связана с типом. Однако язык PHP является гораздо менее формальным. В нем нет необходимости явно определять тип переменных. Интерпретатор PHP самостоятельно проверяет значение, присвоенное некоторой переменной, и после этого связывает с ней соответствующий тип данных. В общем случае это является очень полезным. При этом следует отметить, что процедура автоматического определения типа переменной работает достаточно точно.

Истина или ложь: булевы значения

Переменные логического типа могут содержать два значения: TRUE (истина) и FALSE (ложь). В основном они используются в условных выражениях. Например, результатом обработки выражения `$a>$b` будет либо TRUE (истина), либо FALSE (ложь).

В языке PHP ложными считаются следующие значения.

- строка `"FALSE"` (состоящая из символов как в верхнем, так и в нижнем регистрах);
- целое значение `0`;
- значение с плавающей точкой `0.0`;
- пустая строка;
- строка, содержащая единственный символ `"0"`;
- константа NULL.

Все остальные значения соответствуют логическому значению TRUE. При выводе булевской переменной с помощью функции `echo` отобразится пустая строка, если в этой переменной содержится значение FALSE, и `1` — в противном случае. Логические значения зачастую применяются в качестве возвращаемых значений функций. Это позволяет определить, успешно ли было завершено ее выполнение. Вопросы совместного использования функций и логических переменных более подробно рассматриваются в главе 8.

По мере необходимости в языке PHP автоматически выполняется и преобразование типов. Например, в следующем фрагменте требуемое преобразование выполняется без малейших проблем:

```
$firstNumber = 1;   # Хранится как целое
$secondNumber = 1.1;# Хранится как число с плавающей точкой
$sum = $firstNumber+$secondNumber;
```

С формальной точки зрения, третье выражение лишено смысла, поскольку в нем используются операнды различных типов. Однако интерпретатор PHP выполняет преобразование целого значения в значение типа `float`, и суммирование двух переменных происходит без каких бы то ни было проблем. Следует заметить, что вся эта процедура выполняется автоматически.

Приведение типов

Иногда интерпретатору PHP не удается правильно определить тип переменной. Тогда при ее использовании будет выдана ошибка об использовании неверного типа. В этом случае нужно самостоятельно определить тип переменной. Такая операция называется *приведением типов* (type casting). Явно задать тип можно следующим образом:

```
$newint = (int)$var1;
$newfloat = (float)$var1;
$newstring = (string)$var1;
```

Значение переменной, расположенной справа от символа равенства, присваивается переменной с указанным типом, находящейся слева. Так, значение `$var1` присваивается переменной `$newint` с типом `integer` (`(int)`).

При приведении типов будьте очень внимательны. Иногда это может привести к непредсказуемым результатам. Например, при преобразовании действительного значения в целое теряется дробная часть. Например, если в переменной `$number` содержится значение `1.8`, то после его преобразования в целое — `$newnumber = (int)$number` — в переменной `$newnumber` будет содержаться значение `1`.

Определить тип переменной можно с помощью функции

```
var_dump($myvariable);
```

Например, следующее выражение позволяет определить тип переменной $checkvar:

```
var_dump($checkvar);
```

В качестве результата будет получено выражение int(27), что свидетельствует о том, что в переменной $checkvar содержится целочисленное значение 27.

Работа с числами

Типы данных float и integer являются числовыми. Инициализацию переменных этих типов можно осуществить таким образом:

```
$intvar =3;
$floatvar =9.3;
```

При этом интерпретатор PHP автоматически разместит заданные значения в оперативной памяти с учетом их типа.

Выполнение математических операций

Язык PHP позволяет выполнять над числовыми переменными различные операции. При этом необходимо задать два операнда и соответствующий символ математической операции. Например, операцию сложения (+) можно выполнить следующим образом:

```
1+2
```

То же самое можно осуществить и с переменными:

```
$var1+$var2;
```

При использовании чисел в математических операциях их не следует заключать в кавычки. В противном случае они будут рассматриваться как строковые переменные, над которыми нельзя выполнять необходимые арифметические операции. Однако в отличие от других языков программирования в PHP по необходимости строки автоматически преобразуются в числовой формат. Например:

```
$var1 = "1";
$var2 = 2;
$total = $var1+$var2;
```

Формально переменные $var1 и $var2 просуммировать нельзя, поскольку в $var1 содержится текстовая строка. Однако при обработке этого выражения интерпретатор PHP автоматически преобразует строку "1" в числовое значение 1 и корректно выполнит эту операцию.

В следующем фрагменте также выполняется автоматическое преобразование текстовой строки в число, однако конечный результат уже не так очевиден:

```
$var1 = "x";
$var2 = 2;
$total = $var1+$var2;
```

Поскольку PHP не может преобразовать символ "x" в числовое значение, то при выполнении сложения вместо него используется значение 0. Таким образом, результат сложения переменных $var1 и $var2 будет равен 2. Преобразование типов далеко не всегда приводит к получению желаемых результатов. Безусловно, автоматическое приведение типов является очень удобным и позволяет сэкономить массу усилий. Однако нужно соблюдать осторожность, поскольку оно может привести к получению непредвиденных результатов, как показано в предыдущем примере.

Иногда интерпретатору PHP не удается корректно обработать выражения, которые понятны человеку. Например:

```
$var1 = "2,000";
$var2 = 2;
$total = $var1+$var2;
```

Хотя человеку и понятно назначение запятой в значении первой переменной, интерпретатору PHP — нет. Он преобразует строку `"2,000"` в числовое значение `2`, и в результате после выполнения сложения в переменной `$total` будет содержаться значение `4`.

В табл. 5.1 приведены основные математические операции языка PHP.

Таблица 5.1. Математические операции PHP

Операция	Описание
+	Сложение двух чисел
-	Вычитание второго числа из первого
*	Умножение двух чисел
/	Деление первого числа на второе
%	Остаток от деления (или *деление по модулю*). Например, в результате вычисления выражения `$a = 13%4` в переменной `$a` будет содержаться значение `1`

Порядок выполнения операций

Несколько математических операций можно выполнять одновременно. Например, в следующем выражении используется сразу три операции:

```
$total = 1+2*3+1;
```

Порядок выполнения операций очень важен, поскольку от этого зависит конечный результат. В языке PHP сначала выполняются операции умножения и деления и лишь потом сложение и вычитание. При равном приоритете операций действия выполняются слева направо. Так, в приведенном выше примере значение переменной `$total` становится равным `8`.

```
$total = 1+2*3+1 #сначала выполняется умножение
$total = 1+6+1   #затем - левая операция сложения
$total = 7+1     #затем - сложение справа
$total = 8
```

Изменить последовательность выполнения арифметических операций можно с помощью круглых скобок. Тогда математические операции в скобках будут выполняться в первую очередь. Например, предыдущее выражение можно переписать в следующем виде:

```
$total = (1+2)*3+1;
```

После выполнения всех математических преобразований в переменной `$total` будет содержаться значение `10`:

```
$total = (1+2)*3+1 #сначала выполняется сложение в скобках
$total = 3*3+1     #затем умножение
$result = 9+1      #и наконец сложение
$result = 10
```

Порядок вычислений в скобках соответствует общим правилам выполнения арифметических операций. Например, в выражении `(3+2*5)` сначала выполняется умножение, а затем — сложение. Естественно, внутри скобок можно использовать другие скобки и таким образом изменять последовательность выполнения операций.

Лучше руководствоваться поговоркой "Семь раз отмерь — один раз отрежь" и использовать скобки во всех случаях, когда арифметическое выражение выглядит неоднозначным.

Инкрементирование и декрементирование

Увеличить значение переменной на 1 можно следующим образом:

```
$counter = $counter+1;
```

Однако для этого можно воспользоваться и сокращенной конструкцией языка PHP:

```
$counter++;
```

Например, рассмотрим выражения

```
$counter=0;
$counter++;
echo $counter;
```

В результате в переменной `$counter` будет содержаться значение 1 (что и будет выведено на экран). Точно так же можно выполнить и вычитание:

```
$counter--;
```

Для увеличения значения переменной на 1 можно воспользоваться еще одним выражением: `"+=1"`. С помощью такой конструкции значение переменной можно изменить произвольным образом. Например:

```
$counter+=2;
$counter-=3;
$counter*=2;
$counter/=3;
```

В приведенных примерах выполняется увеличение значения переменной `$counter` на 2, его уменьшение на 3, умножение на 2 и деление на 3 соответственно.

Использование встроенных математических функций

Более сложные математические вычисления можно выполнять с помощью встроенных функций. (Функции более подробно рассматриваются в главе 8.) Например, если нужно найти квадратный корень некоторого значения, нет необходимости создавать новую функцию, так как она уже существует. Например:

```
$rootvar = sqrt(91);
$rootvar = sqrt($number);
```

В первом выражении вычисляется квадратный корень числа 91, а во втором — значения, содержащегося в переменной `$number`.

Для округления действительного числа к ближайшему большему целому можно воспользоваться функцией

```
$upnumber = ceil(27.63);
```

Результатом этого выражения будет 28. Существует аналогичная функция, которая округляет действительное значение к ближайшему меньшему целому. Например:

```
$downnumber = floor(27.63);
```

В этом случае функция `floor()` вернет значение 27.

В языке PHP определено и много других математических функций: для выполнения простых операций, например поиска максимума, минимума или генерирования случайных чисел, или более сложных — вычисления синуса, тангенса или преобразования чисел в двоичный или восьмеричный формат. Перечень математических функций вы найдете в приложении Б.

Форматирование чисел для вывода

Зачастую числа необходимо отображать в каком-то определенном формате, например использовать запятую для разделения тысяч или использовать два знака после запятой для обозначения

денежной суммы. Однако при использовании PHP числа хранятся и отображаются в наиболее эффективном формате. Так, если в переменной содержится число `10.00`, он будет выведено как `10`. Если же нужно вывести это число в другом виде, придется явно указать соответствующий формат.

Для определения формата в PHP предназначена функция `number_format()` с синтаксисом

`number_format(`*`число`*`, `*`количество_десятичных_знаков`*`, "`*`разделитель1`*`", "`*`разделитель2`*`")`

и со следующими параметрами:

- ✓ *число*. Форматируемое число, которое является обязательным аргументом.
- ✓ *количество_десятичных_знаков*. Определяет количество знаков после запятой. Если этот параметр отсутствует, то по умолчанию его значение равно `0` и, таким образом, *число* округляется до ближайшего целого числа (т.е. отображается без дробной части). Если используются аргументы *разделитель1* и *разделитель2*, то *количество_десятичных_знаков* является обязательным параметром.
- ✓ *разделитель1*. Определяет символ, который будет использоваться в качестве символа-разделителя целой и дробной частей. По умолчанию таким символом является точка. Параметр *разделитель1* является обязательным, если используется параметр *разделитель2*.
- ✓ *разделитель2*. Определяет символ, который будет разделителем в целой части числа. По умолчанию в качестве такого символа используется запятая.

В табл. 5.2 приведено несколько примеров использования функции `number_format()`.

Таблица 5.2. Примеры использования функции `number_format()`

`$number`	Формат	Результат
12321	`number_format($number)`	12,321
12321.66	`number_format($number, 2)`	12,321.66
12321.66	`number_format($number)`	12,322
12321.6	`number_format($number, 3)`	12,321.600
12321	`number_format($number, 0, ".", ".")`	12.321
12321.66	`number_format($number, 2, ".", "")`	12321.66

После форматирования число конвертируется в текстовую строку. Поэтому все арифметические операции необходимо выполнить до форматирования.

Для более сложного форматирования в языке PHP предназначены функции `printf()` и `sprintf()`.

- ✓ Функция `printf()` позволяет напрямую выводить число в заданном формате.
- ✓ Функция `sprintf()` сохраняет форматируемое число в переменной.

Функции `printf()` и `sprintf()` одновременно позволяют форматировать как строки, так и числовые значения. Более подробно эти функции рассматриваются ниже, в разделе "Форматирование текстовых строк".

Работа со строками символов

К *символам* (character) относятся буквы, числа и знаки пунктуации. *Строка символов* (или *текстовая строка*) (character string) является последовательностью символов. Если числа используются в качестве символов, они сохраняются так же, как и буквы, т.е. над ними невозможно выполнять арифметические операции. Например, номер телефона обычно хранится как строка, поскольку над ним не нужно выполнять никаких математических действий.

Для присваивания переменной строкового значения используются одинарные или двойные кавычки. Это позволяет указать интерпретатору PHP начало и конец последовательности символов. Например, следующие два выражения идентичны:

```
$string = "Здравствуй, мир!";
$string = 'Здравствуй, мир!';
```

Работа с длинными текстовыми строками

В языке PHP имеется *heredoc*-механизм, позволяющий сохранять в переменных длинные последовательности символов, которые могут занимать несколько строк. Этот механизм позволяет указать начало и конец строки символов с использованием следующего синтаксиса:

$*имя_переменной* = <<<*МЕТКА*

текст

МЕТКА;

Здесь *МЕТКА* представляет собой произвольное имя. Если в переменной $*имя_переменной* необходимо сохранить некоторый текст, его нужно заключить между метками *МЕТКА*. Тогда при обработке этого выражения интерпретатор PHP сможет определить, где находится начало и конец строки, и разместит *текст* в переменной $*имя_переменной*.

В строке, созданной с помощью *heredoc*-механизма, могут содержаться переменные и специальные символы, как и в обычной строке в двойных кавычках. (Такие строки более подробно будут рассматриваться ниже, в разделе "Сравнение строк в одинарных и двойных кавычках".)

Ниже приведен пример создания строки с использованием *heredoc*-механизма.

```
$distance = 10;
$herevariable = <<<ENDOFTEXT
Расстояние между
Лос-Анджелесом и Пасаденой
составляет $distance миль.
ENDOFTEXT;
echo $herevariable;
```

При выводе значения переменной `$herevariable` на экран будет получено следующее:

```
Расстояние между
Лос-Анджелесом и Пасаденой
составляет 10 миль.
```

Однако при использовании таких строк нужно соблюдать осторожность. К выбору меток интерпретатор PHP предъявляет достаточно жесткие требования. При первом появлении *МЕТКА* (в данном примере `ENDOFTEXT`) должна находиться в конце первой строки так, чтобы за ней не было ни одного символа, даже пробела. То же самое касается и завершающей метки: за ней не должно быть никаких символов, кроме точки с запятой. Если эти правила не выполняются, символ конца строки не будет распознан, интерпретатор PHP продолжит поиск этого символа в оставшейся части сценария и в конечном счете будет сгенерировано сообщение об ошибке.

Использование в строках специальных символов

В языке PHP определены специальные символы, \n и \t, которые можно использовать в строках. Символ \n является символом перехода на новую строку. Например:

```
$string = "Здравствуй, \nмир";
echo $string;
```

В результате будет выведено следующее:

```
Здравствуй,
мир
```

Символ \t является символом табуляции. Например, использование фрагмента

```
$string = "Строка 1 \n\tстрока 2";
echo $string;
```

приведет к тому, что вторая строка будет выведена с отступом:

```
Строка 1
     строка 2
```

Специальные символы можно использовать только в строках, заключенных в двойные кавычки. При использовании одинарных кавычек с этими символами не будет связано никакого особого смысла. Различия между этими двумя представлениями строк рассматриваются в следующем разделе.

Сравнение строк в одинарных и двойных кавычках

Строки в одинарных и двойных кавычках обрабатываются интерпретатором PHP по-разному.

- ✓ Последовательность символов, заключенная в одинарные кавычки, хранится "как есть", за исключением символа \', который хранится как символ апострофа. (Более подробно этот вопрос рассматривается в следующем разделе.)
- ✓ В строках в двойных кавычках обрабатываются переменные и специальные символы, и лишь после этого может использоваться сама строка.

В следующих примерах проиллюстрированы различия между строками в одинарных и двойных кавычках.

Если имя переменной разместить в двойных кавычках, интерпретатор PHP будет использовать ее значение, а если в одинарных — ее имя. Например:

```
$name = "Сэм";
$output1 = "$name";
$output2 = '$name';
echo $output1;
echo $output2;
```

При выполнении этого выражения будет получен следующий результат:

```
Сэм
$name
```

Аналогично обрабатываются и специальные символы. Если их разместить в строке с двойными кавычками, то специальные символы будут интерпретироваться, а если в строке с оди-нарными кавычками — нет. Эти отличие проиллюстрированы в следующем примере:

```
$string1 = "Строка в \n\tдвойных кавычках";
$string2 = 'Строка в \n\tодинарных кавычках';
```

Первая строка отобразится так:

```
Строка в
     двойных кавычках
```

а вторая — так:

```
Строка в \n\tодинарных кавычках
```

Наличие разных кавычек определяет, каким образом будут обрабатываться другие кавычки в текстовой строке. Например:

```
$number = 10;
$string1 = "В очереди находится '$number' людей.";
$string2 = 'В очереди находится "$number" людей.';
echo $string1, "\n";
echo $string2;
```

При обработке этого выражения будет получен следующий результат:

```
В очереди находятся '10' людей.
В очереди находятся "$number" людей.
```

Обратите внимание, что, несмотря на то, что в строке `$string1` имя переменной `$number` заключено в одинарные кавычки, двойные кавычки приводят к интерпретации ее значения, а не имени. При выводе второй строки происходит обратное. Поскольку вся строка заключена в одинарные кавычки, то она выводится без интерпретации переменных, имена которых включены в нее.

Сокрытие символов

Иногда необходимо, чтобы в строках с двойными кавычками символы использовались "как есть", т.е. без учета их специального назначения. Например, может понадобиться, чтобы символ доллара интерпретировался как обычный символ, а не как первый символ имени переменной. Такого эффекта можно добиться с помощью символа обратной косой черты (backslash) (\). Например:

```
$string = 'Имя переменной — $var1';
$string = "Имя переменной — \$var1";
```

При выводе обе строки будут выглядеть одинаково:

```
Имя переменной — is $var1
```

Пусть, например, необходимо присвоить строковой переменной следующее значение:

```
$string = 'Где находится дом O'Хары?';
echo $string;
```

Это выражение будет интерпретировано неправильно, поскольку после символа `O` следует символ кавычки (`'`), который будет рассматриваться как символ конца строки. При выводе строки будет получен следующий результат:

```
Где находится дом O
```

Поэтому интерпретатору PHP необходимо указать, что символ кавычки нужно отображать как символ апострофа, а не считать его концом строки. Для этого символ обратной косой черты нужно поместить перед символом кавычки, т.е. для корректного отображения строки нужно внести следующие изменения:

```
$string = 'Где находится дом O\'Хары?';
```

Точно так же, если в строке, заключенной в двойные кавычки, необходимо разместить символ ", то перед ним нужно разместить символ \.

Объединение текстовых строк

Процедура объединения текстовых строк называется *конкатенацией* (concatenation). В языке PHP для выполнения конкатенации используется операция точки (`.`). Рассмотрим следующий пример:

```
$string1 = "Здравствуй,";
$string2 = "мир!";
$stringall = $string1.$string2;
echo $stringall;
```

При выполнении этого фрагмента будет получен следующий результат:

```
Здравствуй,мир!
```

Обратите внимание, что в результирующей строке не содержится символ пробела (""), поскольку он отсутствует в обеих исходных строках. Для добавления символа пробела между отдельными словами нужно объединить три строки: две переменные и непосредственно строку с символом пробела. Например:

```
$stringall = $string1." ".$string2;
```

Для добавления символов к существующей строке можно воспользоваться операцией `.=`. Например, предыдущие примеры можно модифицировать таким образом:

```
$stringall = "Здравствуй,";
$stringall .= " мир!";
echo $stringall;
```

В результате получим следующее:

```
Здравствуй, мир!
```

Манипуляция строками

Для обработки текстовых строк язык PHP предоставляет множество встроенных функций. (Более подробно функции рассматриваются в главе 8.) С помощью стандартных функций в строке можно находить подстроки или отдельные символы, заменять часть строки новыми символами, разбивать строку, находить длину строки и т.д.

Зачастую в начале или в конце строки нужно удалить ненужные пробелы. Это можно осуществить следующим образом:

```
$string = trim($string)  # удаляет пробелы в начале и в конце строки
$string = ltrim($string) # удаляет пробелы только в начале строки
$string = rtrim($string) # удаляет пробелы только в конце строки
```

С использованием функции `str_word_count()` строку можно легко разбить на отдельные слова:

```
str_word_count("строка", формат)
```

Параметр *формат* может иметь два значения: `1` или `2`. При использовании значения `1` результатом выполнения функции `str_word_count()` будет простой массив, элементами которого будут отдельные слова. При использовании значения `2` эта функция возвращает ассоциативный массив, ключами которого будут позиции первых символов слов в строке. (Более подробно массивы рассматриваются в главе 6.) Если параметр *формат* отсутствует, возвращаемым значением функции `str_word_count()` будет количество символов в строке. Рассмотрим следующий пример:

```
$string = "Подсчет слов";
$numberOfWords = str_word_count($string);
```

```
$word1 = str_word_count($string, 1);
$word2 = str_word_count($string, 2);
```

При выполнении этого фрагмента будут получены такие результаты:

```
$numberOfWords = 2
$word1[0] = Подсчет
$word1[1] = слов
$word2[0] = Подсчет
$word2[9] = слов
```

Обратите внимание, что первое слово начинается с позиции с `0` (а не с `1`, как многие могли бы предположить), а следующее — с позиции `9`. Все эти вопросы более подробно рассматриваются в главе 6 при обсуждении массивов.

Некоторые полезные функции, которые можно использовать для обработки текстовых строк, приведены в табл. 5.3. Рассматривая примеры, не забывайте о том, что нумерация символов в строке начинается с нуля.

Форматирование текстовых строк

В языке PHP все значения всегда выводятся в строковом формате. Другими словами, результатом функции `echo` является строка, даже если среди ее параметров содержатся числовые значения. Рассмотрим такой пример:

```
$number = 4;
echo "У Салли $number детей.";
```

В результате выполнения этого фрагмента получим следующую строку:

```
У Салли 4 детей.
```

Несмотря на то что в переменной `$number` содержится число `4`, функция `echo` отображает его как часть символьной строки.

Форматирование данных является одним из важных этапов написания сценариев. Однако функция `echo` оказывается недостаточно гибкой. В разделе "Форматирование чисел для вывода" этой главы уже рассматривались возможности функции `number_format()`, связанные с форматированием чисел. Для форматирования строк в языке PHP также имеются дополнительные возможности. Функции `printf()` и `sprintf()` позволяют форматировать строки, числа, а также их произвольную комбинацию.

Функции `printf()` и `sprintf()` имеют следующий общий синтаксис:

```
printf("формат", $имя_переменной1, $имя_переменной2, ...);
$newvar = sprintf("формат", $имя_переменной1, $имя_переменной2, ...);
```

Функция `printf()` предназначена для вывода уже отформатированной строки, а функция `sprintf()` сохраняет ее в переменной. При этом обе функции позволяют обрабатывать комбинацию чисел, строк и значений переменных. Параметр *формат* позволяет задать шаблон, в соответствии с которым будут форматироваться переменные *$имя_переменной*. Например, следующее выражение является абсолютно корректным:

```
$newvar = sprintf("Здравствуй, мир!");
```

В этом выражении не выполняется никакого форматирования, и строка `Здравствуй, мир!` сохраняется в переменной `$newvar`. Однако строковые литералы можно объединять с переменными с текстовыми символами, как показано в следующем примере:

```
$nboys = 3;
$ngirls = 2;
printf("%s парней и %s девушек", $nboys, $ngirls);
```

Таблица 5.3. Функции для обработки текстовых строк

Функция	Описание	Пример	Результат
`str_repeat("`*строка*`", `*n*`)`	Повторяет *строку n* раз	`$x = str_repeat("x", 5);`	`$x = xxxxx`
`str_replace ("`*a*`", "`*b*`", "`*строка*`")`	В *строке* заменяет все фрагменты *a* фрагментами *b*	`$a = "abc abc"; $s = str_replace("b", "i", $a);`	`$s = aic aic`
`strchr ("`*строка*`", "`*символ*`")`	Возвращает часть *строки*, начиная с *символа* и до конца строки	`$str = "aBc abc"; $sub = strchr($str, "b");`	`$sub = bc`
`stristr ("`*строка*`", "`*символ*`")`	Аналогична функции `strchr()`, но не учитывает регистр	`$str = "aBc abc"; $sub = stristr($str, "b");`	`$sub = Bc abc`
`strlen("`*строка*`")`	Возвращает длину *строки*	`$n = strlen("привет");`	`$n = 6`
`strpos ("`*строка*`", "`*подстрока*`")`	Возвращает позицию первого вхождения *подстроки* в *строку*	`$str = "привет"; $n = strpos($str, "и");`	`$n = 2`
`strrchr ("`*строка*`", "`*символ*`")`	Аналогична функции `strchr()`, но осуществляет поиск последнего фрагмента *символа*	`$str = "abc abc"; $sub = strrchr($str, "b");`	`$sub = bc`
`strrev("`*строка*`")`	Возвращает *строку* в обратном порядке	`$n = strrev("abcde");`	`$n = edcba`
`strrpos ("`*строка*`", "`*подстрока*`")`	Возвращает позицию последнего вхождения *подстроки* в *строку*	`$str = "abc abc"; $n = strrpos($str, "bc");`	`$n = 5`
`strtolower("`*строка*`")`	Преобразует все символы *строки* в нижний регистр	`$str = strtolower("ДА");`	`$str = да`
`strtoupper("`*строка*`")`	Преобразует все символы *строки* в верхний регистр	`$str = strtoupper("да");`	`$str = ДА`
`strtr("`*строка*`", "`*стр1*`", "`*стр2*`")`	Заменяет все вхождения подстроки *стр1* в *строке* значением *стр2*	`$str = "aa bb cc"; $new = strtr($str, "bb", "xx");`	`$new = aa xx cc`
`substr ("`*строка*`", `*n1*`, `*n2*`)`	Возвращает часть *строки*, начиная с позиции *n1* и заканчивая *n2*	`$sstr = substr("привет", 2, 4);`	`$sstr = иве`
`substr_count ("`*строка*`", "`*sub*`")`	Возвращает количество вхождений подстроки *sub* в *строку*	`$str = "abc ab abc"; $s = "bc"; $n = substr_count($str, $s);`	`$n = 2`
`substr_replace ("`*s*`", "`*r*`", `*n*`, `*l*`)`	Заменяет в строке `s` фрагмент из `l` символов, начиная с позиции `n`, фрагментом `r`	`$s = "abc abc"; $t = substr_replace($s, "x", 2, 3);`	`$t = abxbc`
`ucfirst("`*строка*`")`	Преобразует первый символ *строки* в верхний регистр	`$str = "a B c"; $str2 = ucfirst($str);`	`$str2 = A B c`
`ucwords("`*строка*`")`	Преобразует первый символ каждого слова *строки* в верхний регистр	`$str = "aa Bb cc"; $str2 = ucwords($str);`	`$str2 = Aa Bb Cc`

Строка %s представляет собой инструкцию форматирования, которая указывает функции printf() на необходимость вставки значения переменной как строки. Таким образом, результатом выполнения приведенного выше примера будет строка 3 парней и 2 девушек. Символ % сообщает функции printf() о том, что дальше следует инструкция форматирования. Инструкции форматирования имеют следующий общий формат:

%*заполнитель*-*длина*.*точность**тип*

В инструкции форматирования используются следующие параметры.

- ✓ Символ процента (%) является первым символом инструкции форматирования.
- ✓ *Заполнитель*. Символ-заполнитель, используемый для дополнения строки до заданной *длины* (этот параметр описывается ниже). Параметр *заполнитель* может содержать пробел (по умолчанию), 0 или любой другой символ, которому предшествует символ одинарной кавычки. Часто в качестве символа-заполнителя используется значения 01 или 0001.
- ✓ Символ выравнивания (–). При его использовании результат будет выравниваться по левой стороне, в противном случае — по правой (режим по умолчанию).
- ✓ *Длина* определяет ширину выводимого значения. Если количество выводимых символов меньше *длины*, то недостающее пространство заполняется *символом-заполнителем*. Например, если *длина* равна 5, *заполнитель* — 0, а форматируемое значение — 1, то в результате будет получена строка 00001.
- ✓ .*точность*. Определяет, сколько десятичных знаков после точки должно выводиться при форматировании действительных чисел.
- ✓ *тип* значения. Обычно используется строковый тип — s (string). Для вывода чисел с плавающей точкой используется тип f (float).

Ниже приведено несколько примеров использования функции sprintf().

```
$money = 30;
$pet = "Kitten";
$new = sprintf("It costs $%03.2f for a %s.\n", $money, $pet);
$new2 = sprintf("%'.-20s%3.2f", $pet, $money);
echo $new;
echo $new2;
```

В результате выполнения этого фрагмента получим следующий результат.

```
It costs $030.00 for a Kitten.
Kitten.............. 30.00
```

Обратите внимание на то, что для форматирования значения переменной $money используется шаблон 3.2f (три цифры до десятичной точки и две после нее). Этот шаблон используется при формировании обеих строковых переменных, $new и $new2. В то же время в переменной $new значение переменной $money дополняется символом 0, а в $new2 — пробелом.

При добавлении переменной $pet в переменную $new2 использовался шаблон '.-20. Значение 20 определяет количество символов, которые будут использоваться для переменной $pet. Поскольку значение этой переменной Kitten занимает шесть символов, оставшееся пространство дополняется символом-заполнителем ('.), т.е. дополнительно будет выведено 14 точек (.). Инструкция выравнивания (–) определяет, что строка Kitten будет выравниваться по левому краю. В противном случае использовалось бы выравнивание по правому краю.

Зачастую необходимо вывести столбцы чисел. Пусть, например, имеется три числа: 12.3, 1 и 234.55. При использовании функции echo будет получен следующий результат:

```
12.3
1
234.55
```

Даже при использовании функции `number_format()` для задания двух десятичных знаков после точки нужный результат не будет достигнут:

```
12.30
1.00
234.55
```

Для вывода чисел в упорядоченном столбце лучше всего воспользоваться функцией `printf()`:

```
printf("%5.2f\n", $number1);
printf("%5.2f\n", $number2);
printf("%5.2f\n", $number3);
```

Только теперь получен необходимый результат:

```
  12.30
   1.00
 234.55
```

Для вывода числовых значений в данном случае использовался шаблон форматирования `%5.2f\n`. Рассмотрим его составные части более подробно.

- ✓ `%`. Первый символ инструкции форматирования.
- ✓ `5`. Определяет ширину, т.е. количество знаков, которое будет использоваться для вывода чисел. Если количество цифр в числе меньше `5`, то спереди добавляются символы-заполнители (в данном случае пробелы). Поскольку выравнивание справа используется по умолчанию, то символ выравнивания не используется.
- ✓ `2`. Определяет количество знаков после десятичной точки.
- ✓ Тип `f` определяет, что числа будут выводиться как числа с плавающей точкой.
- ✓ `\n`. Символ перевода строки.

Для того чтобы вывести число в денежном формате (со знаком $), можно воспользоваться функцией `sprintf()`. Например:

```
$newvariablename = sprintf("$%.2f", $oldvariablename);
```

В этом выражении форматируется значение переменной `$oldvariablename`, а результат сохраняется в переменной `$newvariablename`. В следующем примере денежное значение выводится в правильном формате:

```
$price = 25;
printf("$%.2f", $price);
```

В результате получим следующее:

```
$25.00
```

Использование даты и времени

Дата и время являются важными элементами сценариев. Поэтому язык PHP предоставляет широкие возможности для их обработки. Компьютер хранит значения даты и времени в формате *timestamp* в секундах. Поскольку этот формат использовать неудобно, в языке PHP имеются различные встроенные функции для его преобразования.

Формат *timestamp* используется в операционной системе Unix и отсчитывает количество секунд, прошедших с 1 января 1970 года, `00:00:00` GMT. Этот формат чрезвычайно удобен при вычислении временной разницы между двумя событиями. Для этого достаточно просто вычесть одно значение из другого.

Форматирование даты

Для форматирования даты чаще всего используется функция `date()`. Она возвращает строковое значение даты и времени в специальном формате. Функция `date()` имеет следующий общий синтаксис:

```
$mydate = date("формат", $timestamp);
```

Параметр `$timestamp` содержит количество секунд. Как описывается в следующем разделе, это значение можно получить с помощью функции `time()` или `mktime()`. Если параметр `$timestamp` не указан, то используется текущая дата и время. Например:

```
$today = date("Y/m/d");
```

Если бы сегодня было 10 марта 2004 года, то в результате было бы выведено следующее значение:

```
2004/03/10
```

Строковый параметр *формат* определяет формат отображения даты и времени. Например, при использовании формата `y-m-d` функция `date()` вернула бы значение `04-3-10`, а при использовании `M.d.Y` — `Mar.10.2004`. В табл. 5.4 приведены различные символы, которые можно использовать в строковом параметре *формат*. (Их полный перечень можно найти на Web-узле `www.php.net`.) Различные элементы шаблона даты могут быть отделены друг от друга дефисом (`-`), точкой (`.`), косой чертой (`/`) или пробелом.

Таблица 5.4. Символы форматирования даты

Символ	Значение	Пример
`M`	Трехбуквенное английское сокращение месяца	`Jan`
`F`	Английское название месяца	`January`
`m`	Месяц (две цифры с нулями)	`02` или `12`
`n`	Месяц (одна-две цифры без нулей)	`1` или `12`
`d`	День месяца (две цифры с нулями)	`01` или `14`
`j`	День месяца (две цифры без нулей)	`3` или `30`
`l`	Английское название дня недели	`Friday`
`D`	Трехбуквенное английское сокращение дня недели	`Fri`
`w`	Порядковое число дня недели, от `0` (воскресенье) до `6` (суббота)	`5`
`Y`	Год (четыре цифры)	`2004`
`y`	Год (две цифры)	`04`
`g`	Часы (12-часовой формат без нулей, от `1` до `12`)	`2` или `10`
`G`	Часы (24-часовой формат без нулей, от `0` до `23`)	`2` или `15`
`h`	Часы (12-часовой формат с нулями, от `01` до `12`)	`01` или `10`
`H`	Часы (24-часовой формат с нулями, от `00` до `23`)	`00` или `23`
`i`	Минуты	`00` или `59`
`s`	Секунды	`00` или `59`

Окончание табл. 5.4

Символ	Значение	Пример
a	до или после полудня: am или pm (в нижнем регистре)	am
A	ДО или ПОСЛЕ полудня: AM или PM (в верхнем регистре)	AM
U	Целое число секунд, прошедших с 1 января 1970 года, 00:00:00 GMT	1056244941

Хранение значений в формате *timestamp*

Для присваивания текущего значения даты и времени в формате *timestamp* предназначена функция `time()`:

```
$today = time();
```

То же самое можно осуществить с помощью выражения

```
$today = strtotime("today");
```

Для получения любого другого значения даты и времени в формате *timestamp* предназначена функция `mktime()`. Она имеет следующий синтаксис:

```
$importantDate = mktime(h, m, s, mo, d, y);
```

где аргумент `h` соответствует часам; `m` — минутам; `s` — секундам; `mo` — месяцу; `d` — дню; `y` — году. Например, для вычисления секунд, прошедших к 15 января 2004 года, необходимо использовать следующее выражение:

```
$importantDate = mktime(0, 0, 0, 1, 15, 2003);
```

Для получения значений в формате *timestamp* можно использовать различные английские ключевые слова или сокращения. Например, приведенное выше выражение можно переписать следующим образом:

```
$importantDate = strtotime("January 15 2003");
```

Функция `strtotime()` может распознавать следующие слова и сокращения.

- ✓ **Название месяцев**: 12 месяцев и соответствующие сокращения.
- ✓ **Название дней недели**: 7 дней и соответствующие сокращения.
- ✓ **Название единиц времени**: `year` (год), `month` (месяц), `fortnight` (две недели), `week` (неделя), `day` (день), `hour` (час), `minute` (минута), `second` (секунда), `am` (до полудня), `pm` (после полудня).
- ✓ **Некоторые английские слова**: `ago` (тому назад), `now` (сейчас), `last` (длиться), `next` (следующий); `this` (этот), `tomorrow` (завтра), `yesterday` (вчера).
- ✓ **Знаки "плюс" и "минус"**.
- ✓ **Все числа**.
- ✓ **Временные зоны**: например, gmt (Greenwich Mean Time — среднее время по Гринвичу) или pdt (Pacific Daylight Time — дневное тихоокеанское время).

Слова и сокращения можно по-разному комбинировать. Например:

```
$importantDate = strtotime("tomorrow");         # 24 часа от сегодня
$importantDate = strtotime("now + 24 hours");
$importantDate = strtotime("last Saturday");
$importantDate = strtotime("8pm + 3 days");
$importantDate = strtotime("2 weeks ago");      # две недели назад
$importantDate = strtotime("next year gmt");    # на один год вперед
$importantDate = strtotime("tomorrow 4am");
```

Для нахождения временной разницы между событиями достаточно просто вычесть одно значение из другого. Например, если переменная `$importantDate` описывает прошедшее событие, то для определения его "давности" соответствующее значение нужно отнять от значения переменной `$today` (которая должна быть определена ранее). Например:

```
$timeSpan = $today-$importantDate;
```

Полученное значение и будет временной разницей в секундах между двумя событиями. Перевести это значение в часы можно следующим образом:

```
$timeSpan = (($today-$importantDate)/60)/60;
```