# **Лабораторная работа №4-5. Работа со строками. Математические операции**

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

## **Простые математические функции**

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

| Простые математические функции PHP | |
| --- | --- |
| Функция | Описание |
| **floor()** | Принимает единственный фактический параметр (как правило, число с плавающей точкой двойной точности) и возвращает наибольшее целое число, которое меньше или равно этому фактическому параметру (округление в меньшую сторону) |
| **ceil()** | Имя этой функции представляет собой сокращение от слова ceiling (потолок). Функция принимает единственный фактический параметр (как правило, число с плавающей точкой) и возвращает наименьшее целое число, которое больше или равно этому фактическому параметру (округление в большую сторону) |
| **round()** | Принимает единственный фактический параметр (как правило, число с плавающей точкой двойной точности) и возвращает ближайшее целое число |
| **abs()** | Модуль числа. Если единственный числовой фактический параметр имеет отрицательное значение, то функция возвращает соответствующее положительное число; если фактический параметр является положительным, то функция возвращает сам фактический параметр |
| **min()** | Принимает любое количество числовых фактических параметров (но не менее одного) и возвращает наименьшее из всех значений фактических параметров |
| **max()** | Принимает любое количество числовых фактических параметров (но не менее одного) и возвращает наибольшее из всех значений фактических параметров |
| **sqrt()** | Вычисляет квадратный корень |

<?php

var\_dump (abs(-3));

var\_dump (round(2.7));

var\_dump (ceil(2.3));

var\_dump (floor(3.9));

$result = min(2, abs(-3), -3,ceil(2.3) );

echo 'минимальное значение= '.$result;

$result = max(2, abs(-3), -3,ceil(2.3) );

echo "<br>";

echo 'максимальное значение = '.$result;

?>

### **Выработка случайных чисел**

В языке PHP применяются два генератора случайных чисел (вызываемых соответственно с помощью функций **rand()** и **mt\_rand()**).

С каждым из этих генераторов связаны по три функции одинакового назначения: функция задания начального значения (**srand()** и **mt\_srand()**), сама функция получения случайного числа и функция, осуществляющая выборку наибольшего целого числа, которое может быть возвращено генератором ((**getrandmax()** и **mt\_getrandmax()**)).

Функции **getrandmax()** и **mt\_getrandmax()** возвращают значение наибольшего числа, которое может быть возвращено функцией **rand()** или **mt\_rand()**, на платформах Windows это значение ограничено величиной 32768.

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

<?php

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

{

echo rand();

echo "<br>";

}

?>

<?php

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

{

echo mt\_rand();

echo "<br>";

}

?>

При использовании некоторых версий PHP для некоторых платформ создается впечатление, что функции **rand()** и **mt\_rand()** вырабатывают на первый взгляд вполне приемлемые случайные числа, даже без предварительного задания начального значения. Но такому впечатлению не следует доверять. Во-первых, программы, в которых используются функции выработки случайных чисел без задания начального значения, невозможно легко переносить на другие платформы, а, во-вторых, надежная работа указанных функций без задания начального значения не гарантируется.

Типичный способ задания начального значения для любого из генераторов случайных чисел PHP (с использованием функции **mt\_srand()** или **srand()**) заключается в следующем. В этом операторе задается начальное значение генератора, равное количеству микросекунд, истекших к данному времени с момента отсчета последней целой секунды. (Приведение типа к типу float в этом операторе действительно необходимо, поскольку функция microtime() возвращает строку, которая рассматривается как целое число в операции умножения, но не в операции передачи параметров в функцию).

Пример:

<?php

mt\_srand((float)(microtime()\*1000000));

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

{

echo mt\_rand();

echo "<br>";

}

?>

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

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

Пример:

Допустим нам нужно сгенерировать случайное число от 100.0 до 120.0

<?php

mt\_srand((float)(microtime()\*1000000));

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

{

$random = 100.0 + 20.0 \* mt\_rand() / mt\_getrandmax();

echo $random;

echo "<br>";

}

?>

Выработка целых чисел (100 - 200);

<?php

mt\_srand((float)(microtime()\*1000000));

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

{

$random = 100.0 + 100.0 \* mt\_rand() / mt\_getrandmax();

echo (int)$random;

echo "<br>";

}

?>

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

### **Математические константы**

В PHP версии 4.0 существовала только одна математическая константа, описанная в документации – M\_PI (значение числа π, представленное в виде числа с плавающей точкой двойной точности). А начиная с версии PHP 4.0.2 было введено много новых констант. Большинство этих новых констант относились к числу π (или к кратным ему значениям), числу e (или к кратным ему значениям), а также к квадратным корням; кроме того, некоторые константы относились к другим типам. Но в следующих выпусках по ряду причин список констант снова сократился до сравнительно небольшого количества заранее заданных математических констант.

| Математические константы PHP | |
| --- | --- |
| Константа | Описание |
| ***M\_PI*** | π |
| ***M\_PI\_2*** | π / 2 |
| ***M\_PI\_4*** | π / 4 |
| ***M\_1\_PI*** | 1 / π |
| ***M\_2\_PI*** | 2 / π |
| ***M\_2\_SQRTPI*** | 2 / sqrt(π) |
| ***M\_E*** | e |
| ***M\_SQRT2*** | sqrt(2) |
| ***M\_SQRT1\_2*** | 1 / sqrt(2) |
| ***M\_LOG2E*** | log2(e) |
| ***M\_LOG10E*** | lg(e) |
| ***M\_LN2*** | loge(2) |
|  | loge(10) |

### **Проверка формата чисел**

В языке PHP предусмотрен ряд функций, позволяющих выполнять проверку правильности представления чисел. Несмотря на то что в языке PHP отсутствует строгий контроль типов, рекомендуется в случае необходимости применять некоторые из этих проверок в коде, чтобы иметь возможность прогнозировать характеристики полученных результатов, а также выбирать наилучший способ их обработки.

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

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

<?php

var\_dump(is\_numeric(4)); *// true*

var\_dump(is\_numeric(25 - 6)); *// true*

var\_dump(is\_numeric("25")); *// true*

var\_dump(is\_numeric("25 - 6")); *// false*

var\_dump(is\_int(4)); *// true*

var\_dump(is\_int(4.2)); *// false*

var\_dump(is\_int("4")); *// false - данная проверка строже, чем проверка с помощью функции is\_numeric()*

var\_dump(is\_float(4)); *// false*

var\_dump(is\_float(4.0)); *// true*

var\_dump(is\_float(M\_PI)); *// true*

var\_dump(is\_float(M\_PI)); *// true*

?>

### **Преобразование систем счисления**

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

Безусловно, после преобразования чисел из внешнего представления во внутреннее они хранятся в памяти в двоичном формате, а все основные арифметические и математические вычисления осуществляются в самой операционной системе по основанию 2. Кроме того, в языке PHP предусмотрен ряд функций для преобразования чисел из одного основания системы счисления в другое. Общие сведения об этих функциях приведены в таблице ниже:

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

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

<?php

var\_dump (bindec('1011'));

var\_dump (hexdec('FF'));

var\_dump (decbin('101'));

var\_dump (dechex('101'));

var\_dump (decoct('101'));

var\_dump (base\_convert('1011', 2,10));

var\_dump (base\_convert('FF', 16,20));

?>

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

### **Экспоненты и логарифмы**

Язык PHP включает стандартные экспоненциальные и логарифмические функции двух разновидностей — для работы по основанию 10 и основанию е (которые приведены в таблице).

| Экспоненциальные функции | |
| --- | --- |
| Функция | Описание |
| ***pow()*** | Принимает два числовых параметра и возвращает первый параметр, возведенный в степень, равную второму параметру. Значение выражения pow($х, $у) равно xy |
| ***exp()*** | Принимает единственный параметр и возводит число e в степень, равную этому показателю степени. Значение выражения exp($х) равно ex |
| ***log()*** | Функция натурального логарифма (ln). Принимает единственный параметр и возвращает его логарифм по основанию e |
| ***log10()*** | Принимает единственный параметр и возвращает его десятичный логарифм (lg) |

В языке PHP предусмотрена функция **exp()** для возведения числа **e** в указанную степень, но отсутствует функция с одним параметром, с помощью которой можно было бы возвести в указанную степень число 10. Однако вместо этой функции можно использовать функцию **pow()** с двумя параметрами, задавая 10 в качестве первого параметра.

<?php

$a=exp(1);

echo 'a='.$a, '<br>';

$a=pow(10,3);

echo 'a='.$a, '<br>';

$a=log10(100);

echo 'a='.$a, '<br>';

?>

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

<?php

$test = 449;

$test = pow(10, exp(log(log10($test))));

echo "test = $test"; *// test\_449 = 449*

?>

### **Тригонометрические функции**

В языке PHP предусмотрен стандартный набор основных тригонометрических функций, общие сведения о которых приведены в таблице:

| Тригонометрические функции | |
| --- | --- |
| Функция | Описание |
| ***pi()*** | Не принимает параметров и возвращает приближенное значение числа π (3.1415926535898). Может использоваться как взаимозаменяемая с константой M\_PI |
| ***sin()*** | Принимает числовой параметр в радианах и возвращает синус параметра в виде числа с плавающей точкой двойной точности |
| ***cos()*** | Принимает числовой параметр в радианах и возвращает косинус параметра в виде числа с плавающей точкой двойной точности |
| ***tan()*** | Принимает числовой параметр в радианах и возвращает тангенс параметра в виде числа с плавающей точкой двойной точности |
| ***asin()*** | Принимает числовой параметр и возвращает арксинус параметра в радианах. Входные данные должны находиться в пределах от -1 до 1 (получение функцией входных данных, выходящих за пределы этого диапазона, приводит к получению результата NAN). Результаты находятся в диапазоне от -π/2 до π/2 |
| ***acos()*** | Принимает числовой параметр и возвращает арккосинус параметра в радианах. Входные данные должны находиться в пределах от -1 до 1 (получение функцией входных данных, выходящих за пределы этого диапазона, приводит к получению результата NAN. Результаты находятся в диапазоне от 0 до π |
| ***atan()*** | Принимает числовой параметр и возвращает арктангенс параметра в радианах. Результаты находятся в диапазоне от -π/2 до π/2 |
| ***deg2rad()*** | Преобразует значение из градусов в радианы |
| ***rad2deg()*** | Преобразует значение из радианов в градусы |

## **Обработка строк**

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

Существует два типа синтаксиса: простой и сложный. Простой синтаксис более легок и удобен. Он дает возможность обработки переменной, значения массива (**array**) или свойства объекта (**object**).

Сложный синтаксис был введен в PHP 4 и может быть распознан по фигурным скобкам, окружающих выражение.

### **Простой синтаксис**

Если интерпретатор встречает знак доллара ($), он захватывает так много символов, сколько возможно, чтобы сформировать правильное имя переменной. Если вы хотите точно определить конец имени, заключайте имя переменной в фигурные скобки.

<?php  
$beer = 'Heineken';  
echo "$beer's taste is great"; *// работает, "'" это неверный символ для имени переменной*  
echo "He drank some $beers";   *// не работает, 's' это верный символ для имени переменной*  
echo "He drank some ${beer}s"; *// работает*  
echo "He drank some {$beer}s"; *// работает*  
?>

Точно также могут быть обработаны элемент массива (**array**) или свойство объекта (**object**). В индексах массива закрывающая квадратная скобка (**]**) обозначает конец определения индекса. Для свойств объекта применяются те же правила, что и для простых переменных, хотя с ними невозможен трюк, как с переменными.

<?php  
*// Эти примеры специфически об использовании массивов внутри строк.   
// Вне строк всегда заключайте строковые ключи вашего  
// массива в кавычки и не используйте вне строк {скобки}.*  
  
*// Давайте покажем все ошибки*  
error\_reporting(E\_ALL);  
$fruits = array('strawberry' => 'red', 'banana' => 'yellow');  
  
*// Работает, но заметьте, что вне кавычек строки это работает по-другому*  
echo "A banana is $fruits[banana].";  
  
*//Работает*  
echo "A banana is {$fruits['banana']}.";  
  
*// Работает, но PHP, как описано ниже, сначала ищет константу banana.*  
echo "A banana is {$fruits[banana]}.";  
  
*// Не работает, используйте фигурные скобки. Это вызовет ошибку обработки.*  
echo "A banana is $fruits['banana'].";  
  
*// Работает*  
echo "A banana is " . $fruits['banana'] . ".";  
  
*// Работает*  
echo "This square is $square->width meters broad.";  
  
*// Не работает. Для решения см. сложный синтаксис.*  
echo "This square is $square->width00 centimeters broad.";  
?>

Для более сложных задач вы можете использовать сложный синтаксис.

### **Сложный (фигурный) синтаксис**

Данный синтаксис называется сложным не потому что труден в понимании, а потому что позволяет использовать сложные выражения.

Фактически, вы можете включить любое значение, находящееся в пространстве имени в строке с этим синтаксисом. Вы просто записываете выражение таким же образом, как и вне строки, а затем заключаете его в **{** и **}**.

Поскольку вы не можете экранировать '**{**', этот синтаксис будет распознаваться только, когда **$** следует непосредственно за **{**. (Используйте "{\$" или "\{$" чтобы отобразить "{$"). Несколько поясняющих примеров:

<?php  
*// Давайте покажем все ошибки*  
error\_reporting(E\_ALL);  
  
$great = 'fantastic';  
  
*// Не работает, выведет: This is { fantastic}*  
echo "This is { $great}";  
  
*// Работает, выведет: This is fantastic*  
echo "This is {$great}";  
echo "This is ${great}";  
  
*// Работает*  
echo "Этот квадрат шириной {$square->width}00 сантиметров.";  
  
*// Работает*  
echo "Это работает: {$arr[4][3]}";  
  
*// Это неверно по той же причине, что и $foo[bar] неверно вне строки.   
// Другими словами, это по-прежнему будет работать,  
// но поскольку PHP сначала ищет константу foo, это вызовет  
// ошибку уровня E\_NOTICE (неопределенная константа).*  
echo "Это неправильно: {$arr[foo][3]}";  
  
*// Работает. При использовании многомерных массивов, внутри  
// строк всегда используйте фигурные скобки*  
echo "Это работает: {$arr['foo'][3]}";  
  
*// Работает.*  
echo "Это работает: " . $arr['foo'][3];  
  
echo "Вы даже можете записать {$obj->values[3]->name}";  
  
echo "Это значение переменной по имени $name: {${$name}}";  
?>

**Доступ к символу в строке и его изменение**

Символы в строках можно использовать и модифицировать, определив их смещение относительно начала строки, начиная с нуля, в фигурных скобках после строки. Приведем примеры:

<?php  
*// Получение первого символа строки*  
$str = 'Это тест.';  
$first = $str{0};  
  
*// Получение третьего символа строки*  
$third = $str{2};  
  
*// Получение последнего символа строки*  
$str = 'Это все еще тест.';  
$last = $str{strlen($str)-1};  
  
*// Изменение последнего символа строки*  
$str = 'Посмотри на море';  
$str{strlen($str)-1} = 'я';  
?>

### **Строковые функции и операторы**

**Строковые операторы**

### **Конкатенация строк**

В различных языках программирования используются различные операторы конкатенации (объединения) строк. Например, в Pascal используется оператор "+". Использование в PHP оператора "+" для конкатенации строк некорректно: если строки содержат числа, то вместо объединения строк будет выполнена операция сложения двух чисел.

В PHP есть два оператора, выполняющие конкатенацию.

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

Второй – оператор присвоения с конкатенацией, который присоединяет правый аргумент к левому.

Приведем конкретный пример:

<?php  
$a = "Hello ";  
$b = $a . "World!"; *// $b содержит строку "Hello World!" - Это конкатенация*   
  
$a = "Hello ";  
$a .= "World!";     *// $a содержит строку "Hello World!" - Это присвоение с конкатенацией*  
?>

### **Операторы сравнения строк**

Для сравнения строк не рекоммендуется использовать операторы сравнения == и !=, поскольку они требуют преобразования типов. Пример:

<?php  
$x = 0;  
$y = 1;  
if ($x == "") echo "<p>x - пустая строка</p>";  
if ($y == "") echo "<p>y - пустая строка</p>";  
*// Выводит:  
// x - пустая строка*  
?>

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

Примеры сравнения строк:

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

Чтобы избежать путанницы и преобразования типов, рекомендуется пользоваться оператором эквивалентности при сравнении строк. Оператор эквивалентности позволяет всегда корректно сравнивать строки, поскольку сравнивает величины и по значению, и по типу:

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

## **Функции для работы со строками**

Для работы со строками в PHP существует множество полезных функций. Кратко разберем часть функций для работы со строками.

### **Базовые строковые функции**

**strlen(string *$st*)**

Одна из наиболее полезных функций. Возвращает просто длину строки, т. е., сколько символов содержится в ***$st***. Строка может содержать любые символы, в том числе и с нулевым кодом (что запрещено в Си). Пример:

$x = "Hello!";  
echo strlen($x); *// Выводит 6*

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

Пытается найти в строке ***$where*** подстроку (то есть последовательность символов) ***$what*** и в случае успеха возвращает позицию (индекс) этой подстроки в строке. Необязательный параметр ***$fromwhere*** можно задавать, если поиск нужно вести не с начала строки ***$from***, а с какой-то другой позиции. В этом случае следует эту позицию передать в ***$fromwhere***. Если подстроку найти не удалось, функция возвращает **false**. Однако будьте внимательны, проверяя результат вызова **strpos()** на false – используйте для этого только оператор ===. Пример:

echo strpos("Hello","el"); // *Выводит 1*

И еще пример:

if (strpos("Norway","rwa") !== false)

echo "Строка rwa есть в Norway";   
*// При сравнении используйте операторы тождественных сравнений (===) (!==), чтобы избежать проблем с определением типов*

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

Данная функция тоже востребуется очень часто. Ее назначение – возвращать участок строки ***$str***, начиная с позиции ***$start*** и длиной ***$length***.

Если ***$length*** не задана, то подразумевается подстрока от ***$start*** до конца строки ***$str***.

Если ***$start*** больше, чем длина строки, или же значение ***$length*** равно нулю, то возвращается пустая подстрока.

Однако эта функция может делать и еще довольно полезные вещи. К примеру, если мы передадим в ***$start*** отрицательное число, то будет считаться, что это число является индексом подстроки, но только отсчитываемым от конца ***$str*** (например, -1 означает «начиная с последнего символа строки»).

Параметр ***$length***, если он задан, тоже может быть отрицательным. В этом случае последним символом возвращенной подстроки будет символ из ***$str*** с индексом ***$length***, определяемым от конца строки.

Примеры:

$str = "Programmer";  
echo substr($str,0,2); *// Выводит Pr*  
echo substr($str,-3,3); *// Выводит mer*

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

Функция **strcmp()** осуществляет безопасное в двоичной форме сравнение двух строк. Сравнение происходит с учетом регистра.

Возвращает три возможных значения:

**0**, если строки полностью совпадают;

-**1**, если строка ***$str1*** лексикографически меньше ***$str2***;

**1**, если, наоборот, ***$str1*** «больше» ***$str2***. Так как сравнение идет побайтово, то регистр символов влияет на результаты сравнений.

<?php  
$out = strcmp("PHP", "php");  
print ($out);  
?>

**Результат:**

-1

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

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

<?php  
$out = strcasecmp("PHP", "php");  
print ($out);  
?>

**Результат:**

0

## **Функции для работы с блоками текста**

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

**str\_replace(string *$from*, string *$to*, string *$str*)**

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

Эта функция работает значительно быстрее, чем **ereg\_replace()**, которая используется при работе с регулярными выражениями PHP, и ее часто используют, если нет необходимости в каких-то экзотических правилах поиска подстроки. Например, вот так мы можем заместить все символы перевода строки на их HTML эквивалент – тэг <br>:

str\_replace(что менять, на что менять, исходная строка)

$st = str\_replace("\n","<br>\n",$str)

Как видим, то, что в строке ***<br>\n*** тоже есть символ перевода строки, никак не влияет на работу функции, т. е. функция производит лишь однократный проход по строке. Для решения описанной задачи также применима функция **nl2br()**, которая работает чуть быстрее.

Еще пример:

<?php

$txt = "Это [B]важный[/B] код.";

$txt\_new = str\_replace( "[B]", "<strong>", $txt );

$txt\_new = str\_replace( "[/B]", "</strong>", $txt\_new );

echo "Исходная строка: \"$txt\".<br>";

echo "Новая строка: \"$txt\_new\".";

?>

**string nl2br(string *$string*)**

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

<?php  
echo nl2br("Первая строка \n Вторая строка");  
?>

*Результат*:

Первая строка   
Вторая строка

*HTML код будет выглядеть следующим образом:*

Первая строка <br />  
Вторая строка

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

Эта функция, появившаяся в PHP4, оказывается невероятно полезной, например, при форматировании текста письма перед автоматической отправкой его адресату при помощи **mail()**. Она разбивает блок текста ***$str*** на несколько строк, завершаемых символами ***$break***, так, чтобы на одной строке было не более ***$width*** букв. Разбиение происходит по границе слова, так что текст остается читаемым. Возвращается получившаяся строка с символами перевода строки, заданными в ***$break***. Пример использования:

<?php  
$str = "Это текст электронного письма, которое нужно будет отправить адресату...";  
*// Разбиваем текст по 20 символов*  
$str = WordWrap ($str, 20, "<br>");  
echo $str;  
*// Выводит:  
/\* Это текст  
электронного письма,  
которое нужно будет  
отправить  
адресату... \*/*  
?>

**strip\_tags (string *$str* [, string *$allowable\_tags*])**

Еще одна полезная функция для работы со строками. Эта функция удаляет из строки все тэги и возвращает результат. В параметре ***$allowable\_tags*** можно передать тэги, которые не следует удалять из строки. Они должны перечисляться вплотную друг к другу. Примеры:

$stripped = strip\_tags($str); *//Удаляет все html - теги из строки (текста)*  
$stripped = strip\_tags($str, "<head><title>"); *//Удалит все html - теги, кроме html - тегов <head> и <title>*

## **Функции для преобразования символьных переменных в массив и наоборот**

**explode ()**

Разбивает строку с помощью разделителя

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

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

*Список параметров:*[*¶*](http://php.net/manual/ru/function.explode.php#refsect1-function.explode-parameters)

***Delimiter*** – Разделитель.

***String*** – Входная строка.

***limit***

Если аргумент ***limit*** является положительным, возвращаемый массив будет содержать максимум ***limit*** элементов, при этом последний элемент будет содержать остаток строки ***string***.

Если параметр ***limit*** отрицателен, то будут возвращены все компоненты кроме последних – ***limit***.

Если ***limit*** равен нулю, то он расценивается как 1.

В результате возвращается массив ([array](http://php.net/manual/ru/language.types.array.php)) строк ([string](http://php.net/manual/ru/language.types.string.php)), созданный делением параметра ***string*** по границам, указанным параметром ***delimiter***.

***Если delimiter является пустой строкой (""), explode() возвращает FALSE. Если delimiter не содержится в string, и используется отрицательный limit, то будет возвращен пустой массив (***[***array***](http://php.net/manual/ru/language.types.array.php)***), иначе будет возвращен массив, содержащий string.***

<?php

*// Пример 1*

$pizza = "кусок1 кусок2 кусок3 кусок4 кусок5 кусок6";

$pieces = explode(" ", $pizza);

echo $pieces[0]; // кусок1

echo "<br>";

echo $pieces[1]; // кусок2

echo "<br>";

var\_dump ($pieces);

?>

<?php

$data = "foo:\*:1023:1000::/home/foo:/bin/sh";

list($user, $pass, $uid, $gid, $gecos, $home, $shell) = explode(":", $data);

echo $user,"<br>"; *// foo*

echo $pass,"<br>"; *// \**

echo $shell,"<br>"; *// /bin/sh*

?>

**Implode ()**

Объединяет элементы массива в строку

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

**string implode ( array *$pieces* )**

Объединяет элементы массива с помощью строки ***glue***.

По историческим причинам, функции **implode()** можно передавать аргументы в любом порядке, однако для унификации с функцией [**explode()**](http://php.net/manual/ru/function.explode.php) следует использовать документированный порядок аргументов.

Список параметров[¶](http://php.net/manual/ru/function.implode.php#refsect1-function.implode-parameters)

***glue*** – по умолчанию равен пустой строке.

***pieces*** – массив объединяемых строк.

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

<?php

$array = array('имя', 'почта', 'телефон');

$comma\_separated = implode(",", $array);

echo $comma\_separated; // имя,почта,телефон

*// Пустая строка при использовании пустого массива:*

var\_dump(implode('hello', array())); // string(0) ""

?>

## **Функции для работы с отдельными символами**

Как и в других языках программирования, в PHP можно работать с символами строк отдельно.

Обратиться к любому символу строки можно по его индексу:

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

**chr(int *$code*)**

Данная функция возвращает строку, состоящую из символа с кодом ***$code***. Пример:

echo chr(75); *//Выводит K*

**ord($char)**

` Данная функция возвращает код символа $char. Вот пример:

echo ord('A'); *// Выводит 65 - код буквы 'A'*

## **Функции удаления пробелов**

Иногда трудно даже представить, какими могут быть странными пользователи, если дать им в руки клавиатуру и попросить напечатать на ней какое-нибудь слово. Так как клавиша пробела – самая большая, то пользователи имеют обыкновение нажимать ее в самые невероятные моменты.

Этому способствует также и тот факт, что символ с кодом 32, обозначающий пробел, как вы знаете, на экране не виден. Если программа не способна обработать описанную ситуацию, то она, в лучшем случае после тягостного молчания отобразит в браузере что-нибудь типа «неверные входные данные», а в худшем – сделает при этом что-нибудь необратимое.

Между тем, обезопасить себя от паразитных пробелов чрезвычайно просто, и разработчики PHP предоставляют нам для этого ряд специализированных функций. Не волнуйтесь о том, что их применение замедляет программу. Эти функции работают с молниеносной скоростью, а главное, одинаково быстро, независимо от объема переданных им строк.

**trim(string *$str*)**

Возвращает копию ***$str***, только с удаленными ведущими и концевыми пробельными символами. Под пробельными символами здесь и далее подразумевается: пробел " ", символ перевода строки \n, символ возврата каретки \r и символ табуляции \t.

Например, вызов **trim(" test\n ")** вернет строку "***test***". Эта функция используется очень широко. Старайтесь применять ее везде, где есть хоть малейшее подозрение на наличие ошибочных пробелов. Поскольку работает она очень быстро.

В этом примере будут удалены символы «Д» и «о» слева и справа. Будьте внимательны с регистром.

<?php  
echo trim("Доброе утро", "До");  
?>

**Результат:**

брое утр

**ltrim(string *$st*)**

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

В этом примере будут удалены символы «Д» и «о» слева. Будьте внимательны с регистром.

<?php  
echo ltrim("Доброе утро!", "До");  
?>

**Результат:**

брое утро!

**chop(string *$st*)**

Удаляет только концевые пробелы, ведущие не трогает.

В этом примере будет удален только символ «о» справа. Будьте внимательны с регистром.

<?php  
echo chop("Доброе утро", "Ро");  
?>

**Результат:**

Доброе утр

## **Функции преобразования символов**

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

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

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

<?php  
echo strtr ("Привет, мои друзья", "иуяПрвтдзь", "iuaPrvtdzj");  
?>

**Результат:**

Privet, моi druzja

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

URL-кодирование необходимо для передачи данных через интернет. Например, такое кодирование целесообразно, если вы передаете русскоязычную информацию в качестве параметра скрипта. Также подобное кодирование можно выполнить и для файла, чтобы не возникало коллизий из-за отсутствия поддержки 8-битных кодировок некоторыми серверами. Вот эти функции:

**UrlEncode(string *$str*)**

Функция URL-кодирует строку ***$str*** и возвращает результат. Эту функцию удобно применять, если вы, например, хотите динамически сформировать ссылку **<a href=...>** на какой-то сценарий, но не уверены, что его параметры содержат только алфавитно-цифровые символы. В этом случае воспользуйтесь функцией так:

echo "<a href=/script.php?param=".UrlEncode($UserData);

Теперь, даже если переменная ***$UserData*** включает символы =, & или даже пробелы, все равно сценарию будут переданы корректные данные.

**UrlDecode(string *$st*)**

Производит URL-декодирование строки. В принципе, используется значительно реже, чем **UrlEncode()**, потому что PHP и так умеет перекодировать входные данные автоматически.

**RawUrlEncode(string *$st*)**

Почти полностью аналогична **UrlEncode()**, но только пробелы не преобразуются в **+**, как это делается при передаче данных из формы, а воспринимаются как обычные неалфавитно-цифровые символы. Впрочем, этот метод не порождает никаких дополнительных несовместимостей в коде.

**RawUrlDecode(string *$st*)**

Аналогична **UrlDecode()**, но не воспринимает **+** как пробел.

**HtmlSpecialChars**(**string $str**, [flags – название константы ]  )

Это функция, которая обычно используется в комбинации с **echo**. Основное ее назначение – гарантировать, что в выводимой строке ни один участок не будет воспринят как тэг.

Заменяет в строке некоторые символы (такие как амперсант, кавычки и знаки «больше» и «меньше») на их HTML-эквиваленты, так, чтобы они выглядели на странице «самими собой». Самое типичное применение этой функции – формирование параметра **value** в различных элементах формы, чтобы не было никаких проблем с кавычками, или же вывод сообщения в гостевой книге, если вставлять тэги пользователю запрещено.

Константы бывают:

**ENT\_COMPAT** – преобразует двойные кавычки, одинарные кавычки не изменяются.

**ENT\_QUOTES** – преобразует как двойные, так и одинарные кавычки.

**ENT\_NOQUOTES** – оставляет без изменения как двойные, так и одинарные кавычки.

**ENT\_IGNORE** – молча отбрасывает некорректные кодовые последовательности вместо возврата пустой строки. Данная возможность предоставляется в целях обратной совместимости, избегайте ее использования, т.к. она может внести уязвимости в ваш код.

**ENT\_SUBSTITUTE** – заменяет некорреткные кодовые последовательности символом замены Юникода U+FFFD в случае использования UTF-8 и &#FFFD; при использовании другой кодировки, вместо возврата пустой строки.

**ENT\_DISALLOWED** – заменяет кодовые последовательности, несоответствующие указанному типу документа символом замены Юникода U+FFFD в случае использования UTF-8 и &#FFFD; при использовании другой кодировки.

**ENT\_HTML401** – обработка кода в соответствии с HTML 4.01.

**ENT\_XML1** – обработка кода в соответствии с XML 1.

**ENT\_XHTML** – обработка кода в соответствии с XHTML.

**ENT\_HTML5** – обработка кода в соответствии с HTML 5.

**Важный момент**, если параметр не указать, то по-умолчанию будет **ENT\_COMPAT.**

**ENT\_IGNORE** – работает начиная с php 5.3

**ENT\_SUBSTITUTE, ENT\_DISALLOWED, ENT\_HTML401, ENT\_XML1, ENT\_XHTML**и**ENT\_HTML5** — работают начиная с php 5.4.

<?php  
$a = "<a href='http://www.phptown.ru'>Руководство по PHP</a>";  
echo "Строка без обработки : ".$a;  
echo "<br>";  
htmlspecialchars("<a href='phptown.php'>Руководство по PHP</a>", ENT\_QUOTES);   
echo "Строка с обработкой: ".htmlspecialchars($a);  
?>

**Результат:**

Строка без обработки : [**Руководство по PHP**](http://www.phptown.ru/)  
Строка с обработкой: <a href='http://www.phptown.ru'>Руководство по PHP</a>

**Код HTML страницы будет выглядеть следующим образом:**

Строка без обработки : <a href='http://www.phptown.ru'>Руководство по PHP</a><br>Строка с обработкой: &lt;a href='http://www.phptown.ru'&gt;Руководство по PHP&lt;/a&gt;

Еще один **важный момент** функция работает с:

1) & (амперсанд) преобразуется в '&amp;'

2) « (двойная кавычка) преобразуется в '&quot;' в режиме ENT\_NOQUOTES is not set.

3) ' (одиночная кавычка) преобразуется в '&#039;' только в режиме ENT\_QUOTES.

4) < (знак «меньше чем») преобразуется в '&lt;'

5) > (знак «больше чем») преобразуется в '&gt;'

**StripSlashes(string *$str*)**

Заменяет в строке ***$str*** некоторые предваренные слэшем символы на их однокодовые эквиваленты. Это относится к следующим символам: **"**, **'**, **\** и никаким другим.

**AddSlashes(string *$str*)**

Вставляет слэши только перед следующими символами: ', " и \. Функцию очень удобно использовать при вызове **eval()** (эта функция выполняет строку, переданную ей в параметрах, так, как будто имеет дело с небольшой PHP-программой.

<?php  
$out = addslashes("PHP'5");  
print ($out);  
?>

**Результат:**

PHP\'5

## **Функции изменения регистра**

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

**strtolower(string *$str*)**

Преобразует строку в нижний регистр. Возвращает результат перевода.

Надо заметить, что при неправильной настройке локали (это набор правил по переводу символов из одного регистра в другой, переводу даты и времени, денежных единиц и т. д.) функция будет выдавать неправильные результаты при работе с буквами кириллицы.

<?php

$str = 'LOREM IPSUM DOLOR SIT AMET';

echo strtolower($str);

?>

**Результат:**

lorem ipsum dolor sit amet

Для кириллицы лучше использовать функцию **string mb\_strtolower(string *$str* [, string encoding])**, функцией идентична **strtolower()**, только вторым параметром передается кодировка.

<?php

$str = 'НО ЧТОБЫ ВЫ ПОНЯЛИ';

echo mb\_strtolower($str, 'utf-8');

?>

**Результат:**

но чтобы вы поняли

Возможно, в несложных программах, а также если нет уверенности в поддержке соответствующей локали операционной системой, проще будет воспользоваться «ручным» преобразованием символов, задействуя функцию **strtr()**:

$st = strtr($st, "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩљЫЬЭЮЯ", "абвгдеёжзийклмнопрстуфхцчшщъыьэюя");

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

**strtoupper(string *$str*)**

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

<?php

echo strtoupper("Hello WORLD!");

?>

**Результат:**

HELLO WORLD!

## **Установка локали (локальных настроек)**

Локалью будем называть совокупность локальных настроек системы, таких как формат даты и времени, язык, кодировка.

Настройки локали сильно зависят от операционной системы.

Для установки локали используется функция **SetLocale()**:

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

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

**LC\_CTYPE** – активизирует указанную локаль для функций перевода в верхний/нижний регистры;

**LC\_NUMERIC** – активизирует локаль для функций форматирования дробных чисел – а именно, задает разделитель целой и дробной части в числах;

**LC\_TIME** – задает формат вывода даты и времени по умолчанию;

**LC\_ALL** – устанавливает все вышеперечисленные режимы.

Теперь о параметре ***$locale***. Как известно, каждая локаль, установленная в системе, имеет свое уникальное имя, по которому к ней можно обратиться. Именно оно и фиксируется в этом параметре. Однако, есть два важных исключения из этого правила.

Во-первых, если величина ***$locale*** равна пустой строке "", то устанавливается та локаль, которая указана в глобальной переменной окружения с именем, совпадающим с именем категории ***$category*** (или LANG – она практически всегда присутствует в Unix).

Во-вторых, если в этом параметре передается 0, то новая локаль не устанавливается, а просто возвращается имя текущей локали для указанного режима.

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

setlocale('LC\_CTYPE','ru\_SU.KOI8-R');

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

По правде говоря, локаль – вещь довольно непредсказуемая и довольно плохо переносимая между операционными системами. Так что, если ваш сценарий не очень велик, задумайтесь: возможно, лучше будет искать обходной путь, например, используя **strtr()**, а не рассчитывать на локаль.

## **Функции преобразования кодировок**

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

**convert\_cyr\_string(string *$str*, char *$from*, char *$to*);**

Функция переводит строку ***$str*** из кодировки ***$from*** в кодировку ***$to***. Конечно, это имеет смысл только для строк, содержащих «русские» буквы, т. к. латиница во всех кодировках выглядит одинаково. Разумеется, кодировка $from должна совпадать с истинной кодировкой строки, иначе результат получится неверным. Значения ***$from*** и ***$to*** – один символ, определяющий кодировку:

**k** – koi8-r  
**w** – windows-1251  
**i** – iso8859-5  
**a** – x-cp866  
**d** – x-cp866  
**m** – x-mac-cyrillic

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

## **Функции форматных преобразований строк**

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

echo "Привет, $name! Вам $age лет.";

В Си для аналогичных целей используется следующий код:

printf("Привет, %s! Вам %s лет",name,age);

Язык PHP также поддерживает ряд функций, использующих такой же синтаксис, как и их Си-эквиваленты. Бывают случаи, когда их применение дает наиболее красивое и лаконичное решение, хотя это и случается довольно нечасто.

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

Эта функция – аналог функции **sprintf()** в Си. Она возвращает строку, составленную на основе строки форматирования, содержащей некоторые специальные символы, которые будут впоследствии заменены на значения соответствующих переменных из списка аргументов.  
Строка форматирования ***$format*** может включать в себя команды форматирования, предваренные символом **%**. Все остальные символы копируются в выходную строку как есть. Каждый спецификатор формата (то есть символ **%** и следующие за ним команды) соответствует одному и только одному параметру, указанному после параметра ***$format***. Если же нужно поместить в текст **%** как обычный символ, необходимо его удвоить:

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

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

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

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

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

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

**>>>** Наконец, обязательный (заметьте – единственный обязательный!) спецификатор типа величины, которая будет помещена в выходную строку:

**b** – очередной аргумент из списка выводится как двоичное целое число;  
**c** – выводится символ с указанным в аргументе кодом;  
**d** – целое число;  
**f** – число с плавающей точкой;  
**o** – восьмеричное целое число;  
**s** – строка символов;  
**x** – шестнадцатеричное целое число с маленькими буквами a-z;  
**X** – шестнадцатеричное число с большими буквами A—Z.

Вот как можно указать точность представления чисел с плавающей точкой:

$money1 = 68.75;  
$money2 = 54.35;  
$money = $money1 + $money2;  
// echo $money выведет "123.1"...  
$formatted = sprintf ("%01.2f", $money);  
// echo $formatted выведет "123.10"!

Вот пример вывода целого числа, предваренного нужным количеством нулей:

$isodate=sprintf("%04d-%02d-%02d",$year,$month,$day);

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

Делает то же самое, что и **sprintf()**, только результирующая строка не возвращается, а направляется в браузер пользователя.

**number\_format(float *$number*, int *$decimals*, string *$dec\_point*=".", string *$thousands\_sep*=",");**

Эта функция форматирует число с плавающей точкой с разделением его на триады с указанной точностью. Она может быть вызвана с двумя или четырьмя аргументами, но не с тремя!

Параметр ***$decimals*** задает, сколько цифр после запятой должно быть у числа в выходной строке.

Параметр ***$dec\_point*** представляет собой разделитель целой и дробной частей.

Параметр ***$thousands\_sep*** – разделитель триад в числе (если указать на его месте пустую строку, то триады не отделяются друг от друга).

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

## **Хэш-функции**

**md5(string *$str*)**

Возвращает хэш-код строки ***$str***, основанный на алгоритме корпорации RSA Data Security под названием «MD5 Message-Digest Algorithm». Хэш-код – это просто строка, практически уникальная для каждой из строк ***$str***. То есть вероятность того, что две разные строки, переданные в ***$str***, дадут нам одинаковый хэш-код, стремится к нулю.

Если длина строки ***$str*** может достигать нескольких тысяч символов, то ее MD5-код занимает максимум 32 символа.

Для чего нужен хэш-код и, в частности, алгоритм MD5? Например, для проверки паролей на истинность.

Пусть, к примеру, у нас есть система со многими пользователями, каждый из которых имеет свой пароль. Можно, конечно, хранить все эти пароли в обычном виде, или зашифровать их каким-нибудь способом, но тогда велика вероятность того, что в один прекрасный день этот файл с паролями у вас украдут.

Сделаем так: в файле паролей будем хранить не сами пароли, а их (MD5) хэш-коды. При попытке какого-либо пользователя войти в систему, мы вычислим хэш-код только что введенного им пароля и сравним его с тем, который записан у нас в базе данных. Если коды совпадут, значит, все в порядке, а если нет – что ж, извините...

Конечно, при вычислении хэш-кода какая-то часть информации о строке ***$str*** безвозвратно теряется. И именно это позволяет нам не опасаться, что злоумышленник, получивший файл паролей, сможет его когда-нибудь расшифровать. Ведь в нем нет самих паролей, нет даже их каких-то связных частей!

Пример использования алгоритма хеширования MD5:

<?php  
$pass\_a = "MySecret";  
$pass\_b = "MySecret";  
*// Выводим хеш-код строки MySecret ($pass\_a) - исходный пароль*  
echo "<b>Хеш-код исходного пароля '$pass\_a':</b><b style=\"color:green\">".md5($pass\_a)."</b><br>";  
*// Выводим хеш-код строки MySecret ($pass\_b) - верифицируемый пароль*echo "<b>Хеш-код верифицируемого пароля '$pass\_b':</b><b style=\"color:green\">".md5($pass\_b)."</b><br>";  
*// Сравниваем хеш-коды MD5 исходного и верифицируемого пароля*  
echo "<h3>Проверяем истинность введенного пароля:</h3>";  
if (md5($pass\_a)===md5($pass\_b)) echo "<h3 style=\"color:green\">Пароль верный! (Хеш-коды совпадают)</h3>";  
else echo "<h3 style=\"color:red\">Пароль неверный! (Хеш-коды не совпадают)</h3>"  
*// В данной ситуации выводит: Пароль верный! (Хеш-коды совпадают)  
// Попробуйте изменить значение строки $pass\_b :)*  
?>

**crc32(string *$str*)**

Функция **crc32()** вычисляет 32-битную контрольную сумму строки ***$str***. То есть, результат ее работы – 32 битное (4-байтовое) целое число. Эта функция работает гораздо быстрее **md5()**, но в то же время выдает гораздо менее надежные «хэш-коды» для строки.

## **Функции сброса буфера вывода**

**flush()**

Эта функция имеет очень и очень отдаленное отношение к работе со строками, но она еще дальше отстоит от других функций.

Начнем издалека: обычно при использовании **echo** данные не прямо сразу отправляются клиенту, а накапливаются в специальном буфере, чтобы потом транспортироваться большой «пачкой». Так получается быстрее.

Однако, иногда бывает нужно досрочно отправить все данные из буфера пользователю, например, если вы что-то выводите в реальном времени (так зачастую работают чаты). Вот тут-то вам и поможет функция **flush()**, которая отправляет содержимое буфера **echo** в браузер пользователя.

## **Преобразование в строковый тип**

Вы можете преобразовывать значения в строковый тип, используя приведение (string), либо функцию **strval()**.

В выражениях, где необходима строка, преобразование происходит автоматически. Это происходит, когда вы используете функции **echo()** или **print()**, либо когда вы сравниваете значение переменной со строкой.

Булевое (**boolean**) значение TRUE преобразуется в строку "1", а значение FALSE представляется как "" (пустая строка). Этим способом вы можете преобразовывать значения в обе стороны – из булева типа в строковый и наоборот.

Целое (**integer**) или число с плавающей точкой (**float**) преобразуется в строку, представленную числом, состоящим из его цифр (включая показатель степени для чисел с плавающей точкой).

Массивы всегда преобразуются в строку "Array", так что вы не можете отобразить содержимое массива (**array**), используя **echo()** или **print()**, чтобы узнать, что он содержит. Чтобы просмотреть один элемент, вам нужно сделать что-то вроде **echo $arr['foo']**.

Объекты всегда преобразуются в строку "Object". Если вы хотите вывести значение переменной-члена объекта (**object**) с целью отладки, прочтите следующие абзацы. Если вы хотите получить имя класса требуемого объекта, используйте **get\_class()**.

Ресурсы всегда преобразуются в строки со структурой "Resource id #1", где 1 – это уникальный номер ресурса (**resource**), присвоенный ему PHP во время выполнения. Если вы хотите получить тип ресурса, используйте **get\_resource\_type()**.

**NULL** всегда преобразуется в пустую строку.

Вывод массивов, объектов или ресурсов не предоставляет вам никакой полезной информации о самих значениях. Более подходящий способ вывода значений для отладки - использовать функции **print\_r()** и **var\_dump()**.

Вы также можете преобразовывать значения PHP в строки для постоянного хранения. Этот метод называется ***сериализацией*** и может быть выполнен при помощи функции **serialize()**. Кроме того, если в вашей установке PHP есть поддержка **WDDX**, вы можете сериализовать значения PHP в структуры XML.

## **Преобразование строк в числа**

Если строка распознается как числовое значение, результирующее значение и тип определяется так как показано далее.

Строка будет распознана как **float**, если она содержит любой из символов '.', 'e', или 'E'. Иначе она будет определена как целое.

Значение определяется по начальной части строки. Если строка начинается с верного числового значения, будет использовано это значение. Иначе значением будет 0 (ноль). Верное числовое значение – это одна или более цифр (могущих содержать десятичную точку), по желанию предваренных знаком , с последующим необязательным показателем степени. Показатель степени – это 'e' или 'E' с последующими одной или более цифрами.

<?php  
$foo = 1 + "10.5";                // $foo это float (11.5)  
$foo = 1 + "-1.3e3";              // $foo это float (-1299)  
$foo = 1 + "bob-1.3e3";           // $foo это integer (1)  
$foo = 1 + "bob3";                // $foo это integer (1)  
$foo = 1 + "10 Small Pigs";       // $foo это integer (11)  
$foo = 4 + "10.2 Little Piggies"; // $foo это float (14.2)  
$foo = "10.0 pigs " + 1;          // $foo это float (11)  
$foo = "10.0 pigs " + 1.0;        // $foo это float (11)       
?>

Более подробную информацию об этом преобразовании смотрите в разделе о **strtod(3)** документации Unix.