*Лекция 8: Работа со строками*

### **Строки**

Вероятно, читатели примерно представляют, что такое тип данных " *строка* " и как *создать* переменную такого типа. В одной из первых лекций мы приводили три способа задания *строк*: с помощью одинарных кавычек, двойных кавычек и с помощью *heredoc* –синтаксиса. Отмечали мы и основные различия между этими способами. В основном они касаются обработки переменных и *управляющих последовательностей* внутри *строки*.

<?php

echo 'В такой строке НЕ обрабатываются

переменные и большинство

последовательностей';

echo "Здесь переменные и последовательности

обрабатываются";

echo <<<EOT

Здесь тоже обрабатываются как переменные,

так и управляющие последовательности.

И кроме того, можно вводить символы кавычек

без их экранирования обратным слэшем.

EOT;

?>

Пример 8.1. Способы задания строк

Уже не раз, начиная с самой первой лекции, мы использовали функцию *echo* . На самом деле, *echo* – не функция, а языковая конструкция, поэтому использовать при ее вызове круглые скобки не обязательно. *Echo* позволяет выводить на экран *строки*, переданные ей в качестве параметров. Параметров у *echo* может быть сколько угодно. Их разделяют запятыми или объединяют с помощью оператора конкатенации и никогда не заключают в круглые скобки.

<?

echo "Пришел ", "увидел ", "победил ";

// выведет строку "Пришел увидел победил"

// многие предпочитают передавать несколько

// параметров в echo с помощью конкатенации

echo "Пришел " . "увидел " . "победил ";

// тоже выведет строку

// "Пришел увидел победил"

echo ("Пришел ", "увидел ", "победил ");

// выдаст ошибку: unexpected ','

?>

Пример 8.2. Использование функции echo

Существует сокращенный синтаксис для команды *echo* :

<?=строка\_для\_вывода?>

Здесь параметр строка\_для\_вывода содержит *строку*, заданную любым из известных способов, которая должна быть выведена на экран.

Например, такой скрипт выведет на экран красным цветом "Меня зовут Вася":

<? $name="Вася" ?>

<font color=red>Меня зовут <?=$name?></font>

Кроме языковой конструкции *echo* существует ряд функций для вывода *строк*. Это в первую очередь функция *print* и ее разновидности printf, sprintf и т.п.

Функция *print* позволяет выводить на экран только одну *строку* и, как и *echo* , не может быть вызвана с помощью переменных функций, поскольку является языковой конструкцией.

Функция print\_r не относится к *строковым функциям*, как можно было бы подумать. Она отображает информацию о переменной в форме, понятной пользователю.

Функции sprintf и printf обрабатывают переданную им *строку* в соответствии с заданным форматом. Но о них мы говорить не будем. А поговорим о том, как можно осуществлять *поиск в тексте, представленном в виде строки*.

### **Поиск элемента в строке**

Для того чтобы определить, входит ли данная подстрока в состав *строки*, используется функция *strpos()* . Синтаксис *strpos()*такой:

strpos (исходная строка,строка для поиска

[,с какого символа искать])

Она возвращает позицию появления искомой *строки* в исходной *строке* или возвращает логическое false, если вхождение не найдено. Дополнительный аргумент позволяет задавать символ, начиная с которого будет производиться *поиск*. Кроме логического false эта функция может возвращать и другие значения, которые приводятся к false (например, 0 или ""). Поэтому для того, чтобы проверить, найдена ли искомая *строка*, рекомендуют использовать оператор эквивалентности " === ".

<?

$str = "Идея наносить данные на перфокарты

и затем считывать и обрабатывать их

автоматически принадлежала Джону Биллингсу,

а ее техническое решение осуществил Герман

Холлерит. Перфокарта Холлерита оказалась

настолько удачной, что без малейших изменений

просуществовала до наших дней.";

$pos = strpos($str,"Холлерит");

if ($pos !== false) echo "Искомая строка

встречена в позиции номер $pos ";

else echo "Искомая строка не найдена";

/\* заметим, что мы проверяем значение

$pos на эквивалентность с false.

Иначе строка, находящаяся в первой позиции,

не была бы найдена, так как 0

интерпретируется как false. \*/

?>

Пример 8.3. Использование функции strpos()

Если значение параметра строка\_для\_поиска не является *строкой*, то оно преобразуется к целому типу и рассматривается как ASCII-код символа. Чтобы получить ASCII-код любого символа в PHP, можно воспользоваться функцией *ord("символ")*

Например, если мы напишем $pos = *strpos*($str,228); то интерпретатор будет считать, что мы ищем символ " д ". Если добавить эту строчку в приведенный выше пример и вывести результат, то получим сообщение, что искомая *строка* найдена в позиции 1.

Функция, обратная по смыслу ord, – это *chr (код символа)* . Она по ASCII-коду выводит символ, соответствующий этому *коду*.

С помощью функции *strpos* можно найти номер только первого появления *строки* в исходной *строке*. Естественно, есть функции, которые позволяют вычислить номер последнего появления *строки* в исходной *строке*. Это функция *strrpos()* . Ее синтаксис таков:

strrpos (исходная строка, символ для поиска)

В отличие от *strpos()* эта функция позволяет найти позицию последнего появления в *строке* указанного символа.

Бывают ситуации, когда знать позицию, где находится искомая *строка*, необязательно, а нужно просто получить все символы, которые расположены после вхождения этой *строки*. Можно, конечно, воспользоваться и приведенными выше функциями *strpos()* и *strrpos()* , но можно сделать и проще – *выделить подстроку* с помощью предназначенных именно для этого функций.

### **Выделение подстроки**

#### **Функция strstr**

Говоря о *выделении подстроки* из искомой *строки* в языке PHP, в первую очередь стоит отметить функцию *strstr()* :

strstr (исходная строка, строка для поиска)

Она находит первое появление искомой *строки* и возвращает подстроку, начиная с этой искомой *строки* до конца исходной *строки*.

Если *строка* для поиска не найдена, то функция вернет false. Если *строка* для поиска не принадлежит *строковому типу* данных, то она переводится в целое число и рассматривается как *код символа*. Кроме того, эта функция *чувствительна к регистру*, т.е. если мы будем параллельно искать вхождения слов "Идея" и "идея", то результаты будут разными. Вместо *strstr()* можно использовать абсолютно идентичную ей функцию *strchr()* .

**Пример 8.4**. Выделим из *строки*, содержащей название и автора исследования, подстроку, начинающуюся со слова "Название":

<?

$str = "Автор: Иванов Иван (<a

href=mailto:van@mail.ru>написать письмо</a>),

Название: 'Исследование языков

программирования' ";

echo "<b>Исходная строка: </b>",$str;

if (!strstr($str, "Название"))

echo "Строка не найдена<br>";

else echo "<p><b>Полученная подстрока: </b>",

strstr($str, "Название");

?>

Пример 8.4. Использование функции strstr()

В результате получим:

Исходная строка: Автор: Иванов Иван

(написать письмо),

Название: 'Исследование языков

программирования'

Полученная подстрока: Название:

'Исследование языков программирования'

Для реализации регистронезависимого поиска подстроки существует соответствующий аналог этой функции – функция *stristr (исходная строка, искомая строка)* . Действует и используется она точно так же, как и *strstr()* , за исключением того, что регистр, в котором записаны символы искомой *строки*, не играет роли при поиске.

Очевидно, что функция *strstr()* не слишком часто используется – на практике редко бывает нужно получить подстроку, начинающуюся с определенного слова или *строки*. Но в некоторых случаях и она может пригодиться. Кроме того, в PHP есть и более удобные функции для поиска вхождений. Наиболее мощные из них, конечно, связаны с регулярными выражениями. Их мы рассмотрим в одной из последующих лекций.

#### **Функция substr**

Иногда мы не знаем, с каких символов начинается искомая *строка*, но знаем, например, что начинается она с пятого символа и заканчивается за два символа до конца исходной *строки*. Как *выделить подстроку* по такому описанию? Очень просто, с помощью функции *substr()* . Ее синтаксис можно записать следующим образом:

substr (исходная строка,

позиция начального символа [, длина])

Эта функция возвращает часть *строки* длиной, заданной параметром длина, начиная с символа, указанного параметром позиция начального символа. Позиция, с которой начинается выделяемая подстрока, может быть как положительным целым числом, так и отрицательным. В последнем случае отсчет элементов производится с конца *строки*. Если параметр длина опущен, то *substr()*возвращает подстроку от указанного символа и до конца исходной *строки*. Длина выделяемой подстроки тоже может быть задана отрицательным числом. Это означает, что указанное число символов отбрасывается с конца *строки*.

**Пример 8.5**. Допустим, у нас есть фраза, выделенная жирным шрифтом с помощью тега <b> языка HTML. Мы хотим получить эту же фразу, но в обычном стиле. Напишем такую программу:

<?php

$word = "<b>Hello, world!</b>";

echo $word , "<br>";

$pure\_str = substr($word, 3, -4);

/\* выделяем подстроку,

начиная с 3-го символа,

не включая 4 символа с конца строки \*/

echo $pure\_str;

?>

Пример 8.5. Использование функции substr()

В результате работы этого скрипта получим:

**Hello, world!** Hello, world!

На самом деле решить такую задачу можно гораздо проще, с помощью функции *strip\_tags* :

strip\_tags (строка [, допустимые теги])

Эта функция возвращает *строку*, из которой удалены все html и php-теги. С помощью дополнительного аргумента можно задать теги, которые не будут удалены из *строки*. Список из нескольких тегов вводится без каких-либо знаков разделителей. Функция выдает предупреждение, если встречает неправильные или неполные теги.

<?php

$string = "<b>Bold text</b>

<i>Italic text</i>";

$str = strip\_tags($string);

// удаляем все теги из строки

$str1 = strip\_tags($string, '<b>');

// удаляем все теги кроме тега <b>

$str2 = strip\_tags($string, '<i>');

// удаляем все теги кроме тегов <i>

echo $str,"<br>",$str1,"<br>", $str2;

?>

Пример 8.6. Использование функции strip\_tags()

В результате получим:

Bold text Italic text

**Bold text Italic text**

*Bold text Italic text*

Приведем другой пример использования функции *substr()* . Допустим, у нас есть какое-то сообщение с приветствием и подписью автора. Мы хотим удалить сначала приветствие, а потом и подпись, оставив только содержательную часть сообщения.

<?php

$text = "Привет! Сегодня мы изучаем работу

со строками. Автор.";

$no\_hello = substr($text, 8);

// убираем приветствие

$content = substr($text, 8, 38);

// то же самое, что substr($text, 8, -6).

// Убираем подпись.

echo $text, "<br>", $no\_hello,

"<br>", $content;

?>

В результате получим:

Привет! Сегодня мы изучаем работу

со строками. Автор.

Сегодня мы изучаем работу со строками. Автор.

Сегодня мы изучаем работу со строками.

Если нам нужно получить один конкретный символ из *строки*, зная его порядковый номер, то не следует задействовать функции типа *substr* . Можно воспользоваться более простым синтаксисом – записывая номер символа в фигурных скобках после имени строковой переменной. В контексте предыдущего примера букву " р ", расположенную второй по счету, можно получить так:

echo $text{1}; // выведет символ "р"

Заметим, что номером этого символа является число один, а не два, так как нумерация символов *строки* производится начиная с нуля.

Раз уж мы начали говорить о символах в *строке* и их нумерации, то невольно возникает вопрос, сколько всего символов в *строке* и как это вычислить. Число символов в *строке* – это ***длина строки*** . Вычислить *длину строки* можно с помощью функции *strlen (строка)*. Например, *длина строки* "Разработка информационной модели" вычисляется с помощью команды: *strlen* ("Разработка информационной модели"); и равна 32 символам.

Итак, как выделять и находить подстроки, мы рассмотрели. Теперь научимся заменять *строку*, входящую в состав исходной *строки*, на другую *строку* по нашему выбору.

### **Замена вхождения подстроки**

#### **Функция str\_replace**

Для *замены вхождения* подстроки можно использовать функцию *str\_replace()* . Это простая и удобная функция, позволяющая решать множество задач, не требующих особых тонкостей при выборе заменяемой подстроки. Для того чтобы производить *замены* с более сложными условиями, используют механизм регулярных выражений и соответствующие функции ereg\_replace() и preg\_replace(). Синтаксис функции *str\_replace()* такой:

str\_replace(искомое значение,

значение для замены, объект)

Функция *str\_replace()* ищет в рассматриваемом объекте значение и *заменяет* его значением, предназначенным для *замены*. Почему мы говорим здесь не про *строки* для поиска и замены и исходную *строку*, а про значения и объект, в котором происходит *замена*? Дело в том, что начиная с PHP 4.0.5 любой аргумент этой функции может быть массивом.

Если объект, в котором производится поиск и *замена*, является массивом, то эти действия выполняются для каждого элемента массива и в результате возвращается новый массив.

<?php

$greeting = array("Привет", "Привет всем!",

"Привет, дорогая!"); // объект

$new\_greet = str\_replace("Привет",

"Доброе утро", $greeting);

// делаем замену

print\_r($new\_greet);

/\* получим: Array ([0]=>Доброе утро

[1]=>Доброе утро всем!

[2]=>Доброе утро, дорогая!) \*/

?>

Пример 8.7. Использование функции str\_replace()

Если искомое значение и значение для *замены* – массивы, то берется по одному значению из каждого массива и производится их поиск и *замена* в объекте. Если значений для *замены* меньше, чем значений для поиска, то в качестве новых значений используется пустая *строка*.

<?php

$greeting = array("Привет", "Привет всем!",

"Привет, дорогая!","Здравствуйте",

"Здравствуйте, товарищи", "Hi");

// объект

$search = array ("Привет",

"Здравствуйте", "Hi");

// значения, которые будем заменять

$replace = array ("Доброе утро",

"День добрый");

// значения, которыми будем заменять

$new\_greet = str\_replace($search, $replace,

$greeting);

// делаем замену

print\_r($new\_greet);

//выводим полученный массив

?>

Пример 8.8. Использование функции str\_replace(). Вариант 2

В результате получим такой массив:

Array (

[0] => Доброе утро

[1] => Доброе утро всем!

[2] => Доброе утро, дорогая!

[3] => День добрый

[4] => День добрый, товарищи

[5] =>

)

Если значения для поиска – массив, а значение для *замены* – *строка*, то эта *строка* будет использована для *замены* всех найденных значений.

<?php

$greeting = array("Привет", "Привет всем!",

"Привет, дорогая!", "Здравствуйте",

"Здравствуйте, товарищи");

// объект

$search = array ("Привет","Здравствуйте");

// значения, которые будем заменять

$replace = "День добрый";

// значение, которым будем заменять

$new\_greet = str\_replace($search,

$replace, $greeting); // делаем замену

print\_r($new\_greet);

//выводим полученный массив

?>

Пример 8.9. Использование функции str\_replace(). Вариант 3

Получим:

Array (

[0] => День добрый

[1] => День добрый всем!

[2] => День добрый, дорогая!

[3] => День добрый

[4] => День добрый, товарищи

)

Функция *str\_replace()* *чувствительна к регистру*, но существует ее регистронезависимый аналог – функция *str\_ireplace()* . Однако эта функция поддерживается не во всех версиях PHP.

Еще один пример использования функции *str\_replace()* – обработка шаблонов.

Обратимся в очередной раз к описанию какого-либо документа, например статьи. Много раз мы уже создавали форму для ввода подобного описания и даже отображали данные, введенные пользователем в такого рода форму. Но как отображать эти данные, мы описывали непосредственно в коде нашей программы. Теперь мы хотим, чтобы способ отображения данных задавал сам пользователь. Для этого добавим в нашу форму еще один элемент для ввода шаблона.

<h2>Введите описание статьи</h2>

<form action=sbl.php>

<table border=0>

<tr><td>Название </td><td><input

type=text name=title > </td></tr>

<tr><td>Краткое содержание </td><td><input

type=textarea name=description > </td></tr>

<tr><td>Автор </td><td><input

type=text name=author > </td></tr>

<tr><td>Дата публикации </td><td><input

type=text name=published ></td></tr>

<tr><td>Шаблон документа </td><td><textarea

name=shablon ></textarea></td></tr>

</table>

<input type=submit value="Отправить">

</form>

Однако просто поля для ввода шаблона недостаточно. Один человек введет в него одно, другой – другое. Нужно договориться о том, как создавать шаблоны, что можно в них использовать, т.е. нужно придумать язык шаблонов. Например, мы договариваемся, что при создании шаблона можно задействовать любые html-теги, а набор *спецсимволов* вида <!имя\_элемента> определяет значение элемента с именем имя\_элемента. Далее, как обрабатывать такого рода шаблоны? Можно использовать функцию *str\_replace()* :

<?php

$tmpl = $\_GET["shablon"];

/\* шаблон, введенный пользователем.

Например, это может быть такая строка:

"<h1><!title></h1> <p><font

size=-1><!description></font></p><p

align=right><!author><br><!published></p>" \*/

function Show(){

// функция, которая производит замену

// элемента шаблона на его значение

global $tmpl;

foreach($\_GET as $k => $v) {

$tmpl = str\_replace("<!$k>",$v,$tmpl);

}

echo $tmpl;

}

Show();

?>

Как эти файлы выглядят для обычного пользователя? Если мы введем в форму такие данные как показано на [рисунке 8.1](https://www.intuit.ru/studies/courses/42/42/lecture/27189?page=4#image.8.1), то в результате получим:

Первая машина для переписи населения

Идея наносить данные на перфокарты и затем

считывать и обрабатывать их автоматически

принадлежала Джону Биллингсу, а ее

техническое решение осуществил Герман

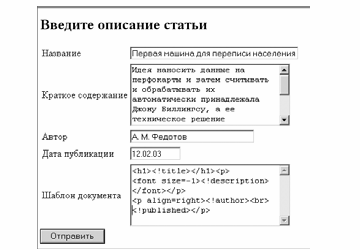
Холлерит. Перфокарта Холлерита оказалась

настолько удачной, что без малейших изменений

просуществовала до наших дней.

А. М. Федотов

12.02.03



**Рис. 8.1.**Форма для ввода описания документа "статья" и шаблона для его отображения

#### **Функция substr\_replace**

Эта функция сочетает в себе свойства двух уже рассмотренных нами функций – функции *str\_replace()* и *substr()* . Ее синтаксис таков:

substr\_replace (исходная строка,

строка для замены,

позиция начального символа [, длина])

Эта функция *заменяет часть строки строкой, предназначенной для замены*. *Заменяется* та часть *строки* (т.е. подстрока), которая начинается с позиции, указанной параметром позиция начального символа. С помощью дополнительного аргумента длина можно ограничить число заменяемых символов. То есть, фактически, мы не указываем конкретно *строку*, которую нужно *заменить*, мы только описываем, где она находится и, возможно, какую длину имеет. В этом отличие функции *substr\_replace()* от *str\_replace()* .

Как и в случае с функцией *substr()* аргументы позиция начального символа и длина могут быть отрицательными. Если позиция начального символа отрицательна, то *замена* производится, начиная с этой позиции относительно конца *строки*. Отрицательная длина задает, сколько символов от конца *строки* не должно быть заменено. Если длина не указывается, то *замена* происходит до конца *строки*.

<?php

$text = "Меня зовут Вася.";

echo "Исходная строка: $text<hr>\n";

/\* Следующие две строки заменят всю

исходную строку строкой 'А меня – Петя' \*/

echo substr\_replace($text, 'А меня – Петя',

0) . "<br>\n";

echo substr\_replace($text, 'А меня – Петя',

0, strlen($text)) . "<br>\n";

// Следующая строка добавит слово 'Привет! '

// в начало исходной строки

echo substr\_replace($text, 'Привет! ',

0, 0) . "<br>\n";

// Следующие две строки заменят имя Вася

// на имя Иван в исходной строке

echo substr\_replace($text, 'Иван', 11,

-1) . "<br>\n";

echo substr\_replace($text, 'Иван', -5,

-1) . "<br>\n";

?>

Пример 8.10. Использование функции substr\_replace()

В результате работы этого скрипта получим:

Исходная строка: Меня зовут Вася.

------------------------------------------

А меня – Петя

А меня – Петя

Привет! Меня зовут Вася.

Меня зовут Иван.

Меня зовут Иван.

### **Разделение и соединение строки**

Очень полезные функции – функция *разделения строки на части* и обратная ей функция *объединения строк* в одну *строку*. Почему очень полезные? Например, если вы динамически генерируете форму по желанию пользователя, можно предложить ему вводить элементы для создания списка выбора, разделяя их каким-нибудь символом. И для того чтобы обработать полученный список значений, как раз и пригодится умение *разбивать строку* на кусочки. Для реализации такого разбиения в PHP можно использовать несколько функций:

explode(разделитель,исходная строка

[,максимальное число элементов])

split (шаблон, исходная строка

[, максимальное число элементов])

preg\_split (шаблон, исходная строка

[, максимальное число элементов

[,флаги]])

Последние две функции работают с регулярными выражениями, поэтому в данной лекции мы их рассматривать не будем. Рассмотрим более простую функцию – *explode()* .

Функция *explode()* *делит исходную строку на подстроки*, каждая из которых отделена от соседней с помощью указанного разделителя, и возвращает массив полученных *строк*. Если задан дополнительный параметр максимальное число элементов, то *число элементов в массиве* будет не больше этого параметра, в последний элемент записывается весь остаток *строки*. Если в качестве разделителя указана пустая *строка* """", то функция *explode()* вернет false. Если символа разделителя в исходной *строке* нет, то возвращается массив с исходной *строкой* без изменений.

**Пример 8.11**. мы хотим создать элемент формы – выпадающий список и значения для этого списка должен ввести пользователь, не знакомый с языком html. Создадим такую форму:

<form action=exp.php>

Введите варианты для выбора автора статьи

через двоеточие (":"):<br>

<input type=text name=author size=40>

<br>

<input type=submit value="Создать элемент">

</form>

Пример 8.11. Использование функции explode()

Скрипт, который будет ее обрабатывать (exp.php), может быть таким:

<?php

$str = $\_GET["author"];

$names = explode(":",$str);

// разбиваем строку введенную,

// пользователем с помощью ":"

$s = "<select name=author>";

// создаем выпадающий список

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

$s .= "<option value=$k>$name";

// добавляем элементы к списку

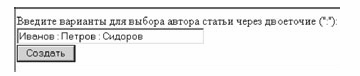
}

$s .= "</select>";

echo $s;

?>

В итоге, если мы введем такую строчку в форму:



**Рис. 8.2.**Ввод значений для создания выпадающего списка

то получим следующий выпадающий список:



**Рис. 8.3.**Выпадающий список, полученный в результате обработки формы

Кроме *разделения строки на части* иногда, наоборот, возникает необходимость *объединения нескольких строк в одно целое*. Функция, предлагаемая для этого языком PHP, называется *implode()* :

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

Эта функция объединяет элементы массива с помощью переданного ей объединяющего элемента (например, запятой). В отличие от функции *explode()* , порядок аргументов в функции *implode()* не имеет значения.

**Пример 8.12**. Допустим, мы храним имя, фамилию и отчество человека по отдельности, а выводить их на странице нужно вместе. Чтобы *соединить их в одну строку*, можно использовать функцию *implode()* :

<?php

$data = array("Иванов","Иван","Иванович");

$str = implode(" ", $data);

echo $str;

?>

Пример 8.12. Использование функции implode()

В результате работы этого скрипта получим *строку*:

Иванов Иван Иванович

У функции *implode()* существует псевдоним – функция join(), т.е. эти две функции отличаются лишь именами.

### **Строки, содержащие html-код**

Достаточно часто мы работаем со *строками*, содержащими html-теги. Если отобразить такую *строку* в браузер с помощью обычных функций отображения данных *echo()* или *print()* , то мы не увидим самих html-тегов, а получим отформатированную в соответствии с этими тегами *строку*. Браузер обрабатывает все html-теги в соответствии со стандартом языка HTML. Иногда нам нужно видеть непосредственно *строку*, без обработки ее браузером. Чтобы этого добиться, нужно перед тем, как *выводить*, применить к ней функцию *htmlspecialchars()* .

Функция *htmlspecialchars (строка [, стиль кавычек [, кодировка]])* переводит специальные символы, такие как "<", ">", "&", """ , "'" в такие сущности языка HTML, как "&lt;", "&gt;", "&amp;", "&quot;", "&#039;" соответственно.

Дополнительный аргумент стиль кавычек определяет, как должны интерпретироваться двойные и одинарные кавычки. Он может иметь одно из трех значений: ENT\_COMPAT, ENT\_QUOTES, ENT\_NOQUOTES. Константа ENT\_COMPAT означает, что двойные кавычки должны быть переведены в спецсимволы, а одинарные должны остаться без изменений. ENT\_QUOTES говорит, что должны конвертироваться и двойные и одинарные кавычки, а ENT\_NOQUOTES оставляет и те и другие кавычки без изменений.

В параметре кодировка могут быть заданы такие кодировки, как UTF-8, ISO-8859-1 и другие (русские кодировки также поддерживаются - см. <http://ru2.php.net/manual/en/function.htmlspecialchars.php>).

<?php

$new = htmlspecialchars("<a

href='mailto:au@mail.ru'>

Написать письмо</a>", ENT\_QUOTES);

echo $new;

/\* наша строка перекодируется в такую:

&lt;a href=&#039;mailto:au@mail.ru&#039;&gt;

Написать письмо&lt;/a&gt; \*/

Пример 8.13. Использование функции htmlspecialchars()

В браузере мы увидим:

<a href='mailto:au@mail.ru'>

Написать письмо</a>

Функция *htmlspecialchars()* перекодирует только наиболее часто используемые спецсимволы. Если необходимо конвертировать все символы в сущности HTML, следует задействовать функцию *htmlentities()* . Русские буквы при использовании этой функции тоже кодируются специальными последовательностями. Например, буква " А " заменяется комбинацией " &Agrave; ". Ее синтаксис и принцип действия аналогичен синтаксису и принципу действия *htmlspecialchars()* .

### ***Заключение***

Итак, мы завершили знакомство с функциями работы со *строками* языка PHP. Конечно же, мы затронули далеко не все существующие функции, а лишь малую часть. Мы изучили функции, позволяющие *найти набор символов в строке*, функции, *заменяющие все вхождения одной строки на другую*, функции *разделения строки на части* и *соединения нескольких строк в одну*, а также рассмотрели функции, позволяющие *выводить на экран строки, содержащие html–код* без их форматирования браузером.