String Theory

By Vikram Vaswani

This article copyright Melonfire 2000−2002. All rights reserved.

Table of Contents

Slice And Dice

If you're anything like me, your acquaintance with PHP's string functions is pretty limited. Sure, you know how to echo output to the screen, and you can even handle things like string concatenation and tokenization. But do you know how to format a string before inserting it into a database, or display decimals in a user−friendly format?

Well, fear not − over the next few pages, I'm going to be taking an in−depth look at the powerful string manipulation tools PHP gives you for free. I'm going to introduce you to the wonders of string concatenation and repetition, string reversal, string comparison, string search−and−replace operations, and string formatting...all in the next ten minutes.

You're probably wondering why you need to know this stuff. For most general development activities, you don't − echo() and print() will do just fine, so long as you're not planning on doing any serious string manipulation. But if you ever find yourself needing to slice and dice strings lengthwise in order to cook them into something completely different, you're going to be glad you read this article. As a PHP developer myself, I was elated to add some new and interesting weapons to my PHP armory while researching this article...and quite distressed that it had taken me so long to find them.

In addition to providing a gentle introduction to PHP programming in general (and PHP string manipulation in particular), this article will offer you a broad overview of PHP's string manipulation capabilities, serving as both a handy reference and a tool to help you write more efficient code. Regardless of whether you're new to PHP or if you've been working with the language for a while, you should find something interesting in here.

Let's get started!

Secret Agent Man

We'll begin right at the top, with some very basic definitions and concepts.

In PHP, the term "string" refers to a sequence of characters. The following are all valid examples of strings:

```
"ciao"

"I ROCK!"

"a long time ago in a galaxy far, far away"
```

String values can be assigned to a variable using the standard assignment operator.

```
<?
$identity = "Spiderman";
?>
```

String values may be enclosed in either double quotes ("") or single quotes(') − the following variable assignments are equivalent"

```
<?
$car = "Porsche";
?>

<?
$car = 'Porsche';
?>
```

String values enclosed in double quotes are automatically parsed for variable names; if variable names are found, they are automatically replaced with the appropriate variable value.

```
<?
$identity = "James Bond";
$car = "BMW";

// this would contain the string "James Bond drives a BMW"
$sentence = "$identity drives a $car";
?>
```

PHP also allows you to create strings which span multiple lines. The original formatting of the string, including newlines and whitespace, is retained when such a string is printed.

```
<?
// multi-line block
$html_output = <<<EOF
<html>
<head></head>
<body>
<ul>
<li>vanilla
<li>chocolate
<li>strawberry
</ul>
</body>
</html>
EOF;
?>
```

The <<< symbol indicates to PHP that what comes next is a multi−line block of text, and should be printed as is right up to the marker "EOF". In PHP−lingo, this is known as "here document" syntax, and it comes in very handy when you need to output a chunk of HTML code, or any other multi−line string.

Strings can be concatenated with the string concatenation operator, represented by a period(.)

```
<?
// set up some string variables
$a = "the";
$b = "games";
$c = "begin";
$d = "now";

// combine them using the concatenation operator

// this returns "the games begin now"
$statement = $a . " " . $b . " " . $c . " " . $d;

// and this returns "begin the games now!"
$command = $c . " " . $a . " " . $b . " " . $d . "!";

// this also returns "begin the games now!"
$command = "$c $a $b $d!";
?>
```

Note that if your string contains quotes, carriage returns or backslashes, it's necessary to escape these special characters with a backslash.

```
<?
// will cause an error due to mismatched quotes
$film = 'America's Sweethearts';

// will be fine
$film = 'America\'s Sweethearts';

// will generate an error
$story = "...and so he said, "backslash me, knave!"";

// will be fine
$story = "...and so he said, \"backslash me, knave!\"";
?>
```

The print() function is used to output a string or string variable.

```
<?
// string
print "Jeepers Creepers";

// string variable
$film = "Jeepers Creepers";
print $film;
?>
```

PHP also offers the echo() construct, which does the same thing.

```
<?
// string
echo "Shakespeare";

// string variable
$author = "Shakespeare";
echo $author;

// combine the two
echo "Despite what critics may say, $author's influence can be
felt even
today";
?>
```

Since displaying variable values is one of the most fundamental things you can do, PHP also offers a shortcut syntax (similar to that offered by JSP) to simplify this task. The following two statements are equivalent:

```
<?
$author = "Shakespeare";
echo $author;
?>

<?=$author?>
```

Running Backwards

With the basics out of the way, let's now turn to some of the other string functions available in PHP. Other than echo() and print(), the three functions you're likely to encounter most often are strlen(), explode() and implode().

The strlen() function returns the length of a particular string, and can come in handy for operations which involve processing every character in a string.

```
<?
$str = "The wild blue fox jumped over the gigantic yellow
pumpkin";

// returns 57
echo strlen($str);
?>
```

The explode() function splits a string into smaller components on the basis of a user−specified pattern, and then returns these elements as an array.

```
<?
$str = "I'm not as think as you stoned I am";

// split into individual words and store in $words[]
$words = explode(" ", $str);
?>
```

This function is particularly handy if you need to take a string containing a list of items (for example, a comma−delimited list) and separate each element of the list for further processing. Here's an example:

```
<?
$str = "Rachel, Monica, Phoebe, Joey, Chandler, Ross";

// split into array
$arr = explode(", ", $str);

// display as list
echo "<ul>";
foreach($arr as $friend)
{
echo "<li>$friend";
}
echo "</ul>";
```

```
?>
```

Obviously, you can also do the reverse − the implode() function creates a single string from all the elements of an array, joining them together with a user−defined separator. Reversing the example above, we have:

```
<?
$arr = array("Rachel", "Monica", "Phoebe", "Joey", "Chandler",
"Ross");

// create string from array
$str = implode(" and ", $arr);

// returns "Rachel and Monica and Phoebe and Joey and Chandler
and Ross are
friends"
echo "$str are friends";
?>
```

The chr() and ord() functions come in handy when converting from ASCII codes to characters and vice−versa. For example,

```
<?
// returns "A"
print chr(65);

// returns 97
print ord("a");
?>
```

In case you need to repeat a string, PHP offers the str_repeat() function, which accepts two arguments − the string to be repeated, and the number of times to repeat it. Here's an example:

```
<?
// returns "Play it again, Sam!" eight times
echo str_repeat("Play it again, Sam!", 8);
?>
```

And if you ever find the need to reverse a string, well, you can always reach for the strrev() function...

```
<?
$str = "Wassup, dood?";
```

```
// reverse string
// $rts now contains ?dood ,pussaW
$rts = strrev($str);

echo "Sorry, you seem to be talking backwards − what does $rts
mean?";
?>
```

I couldn't have put it better myself!

Of Jumping Cows And Purple Pumpkins

Next up, the substr() function. As the name implies, this is the function that allows you to slice and dice strings into smaller strings. Here's what it looks like:

```
substr(string, start, length)
```

where "string" is a string or string variable, "start" is the position to begin slicing at, and "length" is the number of characters to return from "start".

Here's an example which demonstrates how this works:

```
<?
$str = "The cow jumped over the moon, giggling madly as a
purple pumpkin
with fat ears exploded into confetti";

// returns "purple pumpkin with fat ears"
echo substr($str, 50, 28);
?>
```

You can use this function to split a string into smaller chunks of a fixed size,

```
<?
$str = "The cow jumped over the moon, giggling madly as a
purple pumpkin
with fat ears exploded into confetti";

$count = 0;

// length of chunk
$size = 11;

/* returns
The cow jum
ped over th
e moon, gig
gling madly
as a purpl
e pumpkin w
ith fat ear
s exploded
into confet
```

```
ti
*/
while (($size*$count) < strlen($str))
{
$temp = substr($str, ($size*$count), $size);
$count++;
echo "$temp \n";
}
?>
```

or you could take the easy way out and use the built−in chunk_split() function, designed specifically for this purpose.

```
<?
$str = "The cow jumped over the moon, giggling madly as a
purple pumpkin
with fat ears exploded into confetti";

// this is equivalent to the previous example
echo chunk_split($str, 11);
?>
```

You can also use the substr() function to extract a particular character from a string,

```
<?
$str = "The cow jumped over the moon, giggling madly as a
purple pumpkin
with fat ears exploded into confetti";

// returns "j"
echo substr($str, 8, 1);
?>
```

or you can use one of PHP4's cool new features and access a character by specifying its position in the string within curly braces (remember that the first character equates to position 0).

```
<?
$str = "The cow jumped over the moon, giggling madly as a
purple pumpkin
with fat ears exploded into confetti";

// returns "j"
echo $str{8};
```

```
?>
```

Getting Into Position

You can use the case−sensitive strpos() function to locate the first occurrence of a character in a string,

```
<?
$str = "Robin Hood and his band of merry men";

// returns 0
echo strpos($str, "R");
?>
```

and the strrpos() function to locate its last occurrence.

```
<?
$str = "Robin Hood and his band of merry men";

// returns 33
echo strrpos($str, "m");
?>
```

The substr_count() function comes in handy if you need to know how many times a specific patter recurs in a string.

```
<?
$str = "'tis said that the is the most common word in the
English language,
and e is the most common letter";

// returns 4
echo substr_count($str, "the");
?>
```

The strstr() function scans a string for a particular pattern and returns the contents of that string from the point onwards (for a case−insensitive version, try stristr()).

```
<?
$str = "As Mulder keeps saying, the truth is out there";

// returns "the truth is out there"
echo strstr($str, "the");
?>
```

If you need to compare two strings, the strcmp() function performs a case−sensitive binary comparison of two strings, returning a negative value if the first is "less" than the second, a positive value if it's the other ways around, and zero if both strings are "equal". Take a look at a couple of examples to see what this means:

```
<?
// returns −1 because a < s
echo strcmp("apple", "strawberry");

// returns 1 because u > p (s == s)
echo strcmp("superman", "spiderman");

// returns 0
echo strcmp("ape", "ape");
?>
```

You can perform a case−insensitive comparison with the strcasecmp() function, or adopt a different approach with the very cool "natural order" comparison, which compares strings the way humans (rather than computers) would.

Instant Paralysis

You can also search for specific patterns in your strings with regular expressions, via PHP's preg_match() function.

```
<?
$str = "johndoe@somedomain.com";

$pattern =
"/^([a-zA-Z0-9])+([\.a-zA-Z0-9_-])*@([a-zA-Z0-9_-])+(\.[a-zA-Z0-9_-]+)+/";

// returns true
echo preg_match($pattern, $str);
?>
```

You can also use the ereg() function to perform a regular expression match,

```
<?
$str = "yo ho ho and a bottle of gum";

// returns true
echo ereg("bo*", $str);
?>
```

and the ereg_replace() function to perform a search-and-replace operation.

```
<?
$str = "yo ho ho and a bottle of gum";

// returns "yo ho ho and a bottle of rum"
echo ereg_replace("gum", "rum", $str);
?>
```

If you need to perform substitution within a string, PHP also has the str_replace() function, designed specifically to perform find-and-replace operations.

```
<?
$str = "...as Michael dropped into a crouch and came up under
Frank's
punch, he swiveled to the side and kicked Frank in the spine,
immediately
```

```
breaking one of Frank's vertebrae and rendering him paralyzed
for life...";

// returns "...as Michael dropped into a crouch and came up
under John's
punch, he swiveled to the side and kicked John in the spine,
immediately
breaking one of John's vertebrae and rendering him paralyzed
for life..."
echo str_replace("Frank", "John", $str);
?>
```

A Quick Trim

If you're looking to perform a little cosmetic surgery on your strings, a good place to start is the family of trim() functions. The most useful of these, trim(), is constructed specifically to remove whitespace from the beginning and end of a string. This comes in handy if you need to remove whitespace from a value prior to using it elsewhere (a database insert, maybe?)

```
<?
$str = " ever seen a white pigeon?";

// returns "ever seen a white pigeon?"
echo trim($str);
?>
```

It's also a good idea to use the trim() function on data entered into online forms, in order to ensure that your error−checking routines don't miss entries containing only whitespace. Here's an example which illustrates what I mean:

```
<?
$search = " ";

// bad code, will not identify that search string actually
contains nothing
if ($search != "")
{
perform_search();
}

// good code, will account for whitespace−only entries
if (trim($search) != "")
{
perform_search();
}
?>
```

You can also use the ltrim() and rtrim() functions, which remove whitespace from the beginning and end of a string respectively.

The next few string functions come in very handy when adjusting the case of a text string from lower− to upper−case, or vice−versa:

strtolower() − convert string to lower case strtoupper()− convert string to upper case ucfirst() − convert the first character of string to upper case ucwords() − convert the first character of each word in string to upper case

Here's an example:

```
<?
$str = "Something's rotten in the state of Denmark";

// returns "something's rotten in the state of denmark"
echo strtolower($str);

// returns "SOMETHING'S ROTTEN IN THE STATE OF DENMARK"
echo strtoupper($str);

// returns "Something's rotten in the state of Denmark"
echo ucfirst($str);

// returns "Something's Rotten In The State Of Denmark"
echo ucwords($str);
?>
```

You've already used the print() function extensively to display output. However, the print() function doesn't allow you to format output in any significant manner − for example, you can't write 1000 as 1,000 or 1 as 00001. And so the clever PHP developers came up with the sprintf() function, which allows you to define the format in which data is output.

Consider the following example:

```
<?
// returns 1.6666666666667
print(5/3);
?>
```

As you might imagine, that's not very friendly. Ideally, you'd like to display just the "significant digits" of the result. And so, you'd use the sprintf() function:

```
<?
// returns 1.67
echo sprintf("%1.2f", (5/3));
?>
```

A quick word of explanation here: the PHP sprintf() function is very similar to the printf() function that C programmers are used to. In order to format the output, you need to use "field templates", templates which represent the format you'd like to display.

Some common field templates are:

%s string
%d decimal number
%x hexadecimal number
%o octal number
%f float number

You can also combine these field templates with numbers which indicate the number of digits to display − for example, %1.2f implies that PHP should only display two digits after the decimal point. If you'd like the formatted string to have a minimum length, you can tell PHP which character to use for padding by prefixing it with a single quote (').

Here are a few more examples of sprintf() in action:

```
<?
// returns 00003
echo sprintf("%05d", 3);

// returns $25.99
echo sprintf("$%2.2f", 25.99);

// returns ****56
echo sprintf("%'*6d", 56);
?>
```

In addition to the sprintf() function, PHP also offers the strpad() function, which is used for padding strings to a specific length. This function accepts a string or string variable as argument, together with the minimum string length required; a couple of optional arguments allow you to also specify which character to use for padding, and the direction in which padding is to take place.

Here are a couple of examples:

```
<?
$str = "da bomb";

// returns "da bomb "
echo str_pad($str, 10);

// returns "da bomb###"
echo str_pad($str, 10, "#");

// returns "***da bomb"
echo str_pad($str, 10, "*", STR_PAD_LEFT);
?>
```

Finally, the wordwrap() function can be used to break long sentences at a specified length.

```
<?
$str = "It's been ten years since Dr. Hannibal \"The
Cannibal\" Lecter
(Anthony Hopkins) escaped from a maximum−security penitentiary
− ten years
in which he's roamed free, indulging his very specialized
tastes. But out
of sight is very definitely not out of mind − he still haunts
Clarice
Starling (Julianne Moore), now a special agent in the FBI.";

// returns a word−wrapped block of width 50 characters
/*
It's been ten years since Dr. Hannibal "The
Cannibal" Lecter (Anthony Hopkins) escaped from a
maximum−security penitentiary − ten years in which
he's roamed free, indulging his very specialized
tastes. But out of sight is very definitely not
out of mind − he still haunts Clarice Starling
(Julianne Moore), now a special agent in the FBI.
*/
echo wordwrap($str, 50);
?>
```

Working The Web

PHP also comes with a bunch of functions constructed specially for Web development. The first of these is the very cool addslashes() function, which automatically escapes special characters in strings. You should make it a point to run this function on your variables prior to inserting them into a database (or any other application that has trouble with special characters).

```
<?
$str = <<<EOF
When Nicholas "Oz" Oseransky (Matthew Perry) goes over to
introduce himself
to his new neighbour, he's surprised to recognize notorious
mob hit-man,
Jimmy "The Tulip" Tudeski (Bruce Willis), currently in hiding
after
squealing on the Gogolack mob "family" in Chicago.
EOF;

// returns a string with all special characters escaped
/*
When Nicholas \"Oz\" Oseransky (Matthew Perry) goes over to
introduce
himself to his new neighbour, he\'s surprised to recognize
notorious mob
hit-man, Jimmy \"The Tulip\" Tudeski (Bruce Willis), currently
in hiding
after squealing on the Gogolack mob \"family\" in Chicago.
*/
echo addslashes($str);
?>
```

You can reverse the process with the stripslashes() function, which removes all the backslashes and returns a "clean" string.

The htmlentities() and htmlspecialchars() functions automatically convert special symbols (like < and >) into their corresponding HTML representations (< and >). Similarly, the nl2br() function automatically replaces blank lines in a string with the corresponding HTML line break tag

.

```
<?
$str = "if (x < 5 &y > 8) \n { \n self_destruct() \n } \n";

// returns "if (x < 5 && y > 8) <br> { <br> self_destruct()
<br> } <br>"
echo nl2br(htmlentities($str));
```

```
?>
```

The strip_tags() functions works in the opposite manner, finding and removing all HTML and PHP tags that may be embedded within the string.

```
<?
$str = "<table><tr><td>Name</td><td>Price</td></tr></table>";

// returns "NamePrice"
echo strip_tags($str);
?>
```

And that's about all I have. I hope you enjoyed this article, and that it offered you some insight into the massive amount of string processing power at your disposal in PHP4.

For more information on any of the functions listed here, take a look at the PHP manual page on strings at http://www.php.net/manual/en/ref.strings.php ...and until next time, stay healthy!