

PHP Back to Basics #10: String Handling in PHP

In this tutorial, we're going to look at some of the most useful string handling functions that PHP has to offer. PHP has a huge range of string handling functions, which make manipulating text easy, whether it's read from a file or database, or entered by a user.

We'll be looking at some of the functions you'll find most helpful as a series of "How To's", explaining the commands and giving some examples of how they can be used.

Working with Strings

In this section of the tutorial, we'll be looking at commands that allow us to find information about a string.

How to Find the Length of a String – strlen()

To find the length of a string, we use the PHP **strlen()** command. This takes a string as a parameter and then returns the number of characters in the string. This can come in useful in a number of situations, for example:

- Checking that the user has entered at least the required number of characters into a text field, for example in a password field.
- Checking that a text string that will be saved in a database field is not larger than the maximum number of characters that the field can hold.

As an example, try the following code:

```
<?php
$password = "example";
echo "The length of the password is " . strlen($password) . " characters";
?>
```

When this is run, the following output can be seen:

```
The length of the password is 7 characters
```

How to Convert an Array to a String – implode()

Often, it can be useful to convert data stored in an array into a string. This allows you to save the contents of an array in a session variable, or in a single database field. To do this we use the **implode()** command which uses the following format:

```
$result = implode($separator,$array);
```

\$separator is a character which is used to split the array elements in the string, and should be a character which will never appear in the data.

For our next example, try the code below:

```
<?php
$data['name'] = "Gareth";
$data['age'] = "26";
$data['language'] = "PHP";
$result = implode(":",$data);
echo $result;
?>
```

Here, we have set up an array, and we then use the **implode()** command to turn the array into a string, using a colon as a separator. When you run the code the following output is shown:

```
Gareth:26:PHP
```

This can then be saved in a session variable, cookie or database field.

Note however, that in the case of an associative array like the one in the example, the arrays keys are lost, but, as long as you know the order of the data, you can easily reconstruct the array.

How to Convert a String to an Array – explode()

The **explode()** command is the opposite to the **implode()** command above, and splits a string into an array, using a separator to break up the string. The format for the **explode()** command is shown below:

```
$array = explode($separator,$string);
```

You can see the command in action with the following code:

```
<?php
$string = "Gareth:26:PHP";
$array = explode(":",$string);
echo "<pre>";
var_dump($array);
echo "</pre>";
?>
```

When this code is run you will see the following result:

```
array(3) {
  [0]=>
  string(6) "Gareth"
  [1]=>
  string(2) "26"
  [2]=>
  string(3) "PHP"
}
```

And you can see that the string has been split into an array.

How to Replace Part of a String with Another String – str_replace()

Often, you'll want to replace part of a string with another string. For example, you could replace the product code in a string with the product name. The **str_replace()** command uses the format:

```
$result = str_replace($search_text,$replace_text,$string);
```

\$search_text is the text to find, **\$replace_text** is the text to replace it with, and **\$string** is the text in which to perform the replacement.

We can see an example of the **str_replace()** command in action with the code below:

```
<?php
$text = "Woof the cat is 6 years old";
$find = "cat";
$replace = "dog";
$result = str_replace($find,$replace,$text);
echo $result;
?>
```

When this code is run, you'll see the following result:

```
Woof the dog is 6 years old
```

And you can see that the word 'cat' in the original text has been changed to 'dog'.

Finding the Position of a String Inside Another String – strpos()

Sometimes, you may want to check for the position of a string inside another string, so that you can then extract part of the string with the **substr()** command. To do this we use the **strpos()** command, using the format:

```
$result = strpos($string, $find_text);
```

The command returns either Boolean **false** if the text to find is not in the string, or if the find string is found, the first position of the text in the string will be returned.

We can see an example of this with the following code:

```
<?php
$string = "This article is written by Gareth Downes-Powell";
$result = strpos($string,"Gareth");
if($result === FALSE){
    echo "The name Gareth was not found in the text";
} else {
    echo "The name Gareth was found at position " . $result;
}
?>
```

When this code is run the following output can be seen:

```
The name Gareth was found at position 27
```

Finding the Number of Times Text occurs in a String – substr_count()

To find the number of times a piece of text occurs in a string, we can use the **substr_count()** command which uses the format:

```
$result = substr_count($string,$find_text);
```

To show this command in use, try the following code:

```
<?php
$text = "This is a piece of example text";
$find = "is";
$result = substr_count($text,$find);
echo $find . " was found " . $result . " times in the text";
?>
```

Running the code yields:

```
is was found 2 times in the text
```

Notice that 'is' was found twice, as it also occurs in the word 'this'!

Working with User Inputted Strings

In this section, we are going to look at commands that can be used to format a user's input. These commands can be extremely useful, especially when the input is going to be entered into a database.

Removing Extra Spaces from a String – trim()

When you collect input from a user often you'll find that they enter extra spaces at the end of the text. You can easily remove these by using the **trim()** command, as shown below:

```
<?php
$text = "This is an example.   ";
$result = trim($text);
?>
```

You can now store **\$result** in a database without storing extraneous spaces.

Removing HTML and PHP Tags from User Input – strip_tags()

Often, where you allow the user to enter input on a web page, you'll want to remove any HTML or PHP tags they enter in the code that could disrupt your own formatting. To do this you can use the **strip_tags()** command which uses the following format:

```
$result = strip_tags($text[,,$allowed tags]);
```

If you specify the optional parameter **\$allowed_tags** then any tags included will not be stripped from the text. So for example:

```
<?php
$text = "<p>This is an example<br></p>";
$result = trim($text, "<p>");
?>
```

In the above example, the `
` tag will be removed, and the `<p>` tags will be left alone.

Escaping Characters that could Interfere with SQL such as ` - addslashes()

A common problem with entering users data straight into a database is that some characters can interfere with SQL statements such as the ``` character, which, if included in the data, would break the SQL statement. An example of this is if the user entered a name such as O'Brien, which could cause problems when you try to input it into the database.

To counter this, we use the **addslashes()** command to modify the data before it's inserted. The characters that will cause problems won't be stripped, but they will be escaped so they can be inserted correctly.

An example of using the **addslashes()** command is shown in the code below:

```
<?php
$name = "O'Brien";
$newdata = addslashes($name);
?>
```

\$newdata could now be safely used in an SQL statement.

Converting Text Escaped with the addslashes() Command - stripslashes()

The **stripslashes()** command is the opposite of the **addslashes()** command, and will unescape characters such as single quotes which need to have an escape character added when they're entered into the database.

Before you display the data read from the database, you simply use the **stripslashes()** command as shown below:

```
<?php
$name = "O'Brien";
$newdata = stripslashes($name);
?>
```

Formatting a String

In this section, we will be looking at various ways of formatting a string, so that we can output it in different ways.

How to Output the First x Characters of a String – substr()

To read the first x numbers of characters of a string, we can use the **substr()** command. This is extremely useful when you have a long piece of text in a database, for example a news report, and you want to display the first 50 characters for example, as a short link to give the user a taster of the longer text.

The **substr()** command has the following format:

```
$new_string = substr($original_string, $start_position, $length);
```

The **\$original_string** parameter is the string that you want to shorten, **\$start_position** is the position in the string that you want to start taking characters from, and **\$length** is the number of characters you want to extract.

The result is returned in **\$new_string**.

As an example, try the following:

```
<?php
$text = "This is an example piece of text, which we will shorten using the substr()
command";
$short = substr($text,0,14);
echo $short;
?>
```

When this code is run, you'll see the following output:

```
This is an exa
```

In this case, the first 14 characters of the string are displayed, starting from and including the first character (position 0). Although this is fine in a number of cases, you'll see that the word 'example' is split which may not be desirable, and we'll look at a neater version next which won't split in the middle of a word.

How to Output the First x Number of Words, up to a Maximum of y Characters – explode()

We've already looked at the **explode()** command in the section on working with strings, but we'll look at another use for it here. In the code for the **substr()** command above, we saw that although the command displayed the first x number of characters, it can split the text in the middle of a word.

Using the **explode()** command, we can split a text string into individual words, and then display the words making sure that we don't go over a maximum number of characters, so that we can truncate a string but without splitting the string in a middle of a word.

Let's see how this works in practice:

```
<?php
$text = "This is an example piece of text, which we will shorten using the substr()
command";
$split = explode(" ", $text);
$maximumCharacters = 22;

$counter = 0;
foreach($split as $word){
    $tempCount = $counter + strlen($word);
    if($tempCount < $maximumCharacters){
        $short = $short . $word . " ";
        $counter = $counter + strlen($word) + 1;
    } else {
        break;
    }
}
echo $short;
?>
```

First, we set up a variable called **\$maximumCharacters** that the output string cannot be longer than – in this case 22. We then use the **explode()** command to break up the string in **\$text** into individual words by breaking the string each time it finds a space, and the results are held in the array **\$split**.

We then use a **foreach** loop to work through the **\$split** array, so we take one word from the array at a time, which is held in **\$word**. We then find the length of the word in **\$word**, and add it to the value in **\$counter** which keeps track of the total number of characters in the final string.

If this value is less than the maximum number of characters allowed, we add the word to the final string with a space after it, and add the length of the word to the value in **\$counter**.

Finally, we output the new string held in **\$short**.

So when the code is run, you'll see the following output:

```
This is an example
```

So you can see the code has had the desired effect, and the text is split without splitting in the middle of a word. Incidentally, if we had just used the **substr()** command, asking it for 22 characters we would have seen:

```
This is an example pie
```

How to Convert a String to Lower Case – strtolower()

In some cases, you may want to have a string stored or displayed in lower case (that is with no capital letters), even if the original string contained capitals.

This is especially useful when comparing input strings. For example while you may expect, and require, users to enter a string containing the same letters, it's not always reasonable to expect them to always enter the string with the same capitalisation.

By converting both strings you're comparing into a single case (such as lowercase) you can check that the two strings are the same in meaning without capital letters making a difference to the comparison.

To do this we use the **strtolower()** command, as shown in the following code.

```
<?php
$text = "This is an example piece of text. It contains Capital and Lower Case letters";
echo strtolower($text);
?>
```

When this code is run, the following output is displayed.

```
this is an example piece of text. it contains capital and lower case letters
```

And you can see that all capital letters have been changed to lower case.

How to Convert a String to Upper Case – strtoupper()

As the name suggests, **strtoupper()** works in a reverse way to the **strtolower()** command, and changes a string into upper case (that is all capitals).

This can be useful for dealing with, for example, product codes, which may be stored in the database as upper case (say PROD01), but could be entered by a user in lower case (prod01).

Try the following code:

```
<?php
$text = "This is an example piece of text. It contains Capital and Lower Case letters";
echo strtoupper($text);
?>
```

When this code is run you will see the following output:

```
THIS IS AN EXAMPLE PIECE OF TEXT. IT CONTAINS CAPITAL AND LOWER CASE LETTERS
```

And you can see that this time the code now only contains capital letters.

How to Format a Number – number_format()

In many cases you will want to display the result of a calculation, but with the result correctly formatted, to a certain number of decimal places, for example. To do this we use the **number_format()** command, which has the following syntax:

```
$result = number_format($number[,decimal places]);
```

When you use the **number_format()** command without giving the decimal places parameter, the number is formatted in grouped thousands using a comma – for example 123,456.

If you pass the command a number of decimal places, the number will be formatted to that number of places.

We can see the command in action as follows:

```
<?php
$number = 123456;
echo number_format($number);
echo "<br>";
$number = 123456.4343;
echo number_format($number,2);
?>
```

Here we have two examples, one formatting a number with grouped thousands only, and another example with grouped thousands and to two decimal places.

When the code is run you will see the following output:

```
123,456
123,456.43
```

And you can see the number has been nicely formatted.

How to Change New Lines into HTML
 tags – nl2br()

Often, you'll want to change standard line breaks into
 tags so that these line breaks display properly in a browser. An example of when you would use this is if you have data entered by the user in a multi line text box on a form.

If a user enters line breaks in the box to format the text, when you try to display this data on a web page the line breaks will be ignored. To counter this, we use the **nl2br()** command. This will change the line breaks from the new line character (\n) to an HTML
 tag, so the text will display correctly.

As an example, try this:

```
<?php
$text = "This is an example of multiline input.\n This is the second line.\n This is
the the third line";
echo $text;
echo "<br><br>";

$result = nl2br($text);
echo $result;
?>
```

First, we create a string with line breaks. We then display this text and you can see that the line breaks are ignored.

Next, we process the text with the **nl2br()** command and display the result. All is now well!

```
This is an example of multiline input. This is the second line. This is the the third
line

This is an example of multiline input.
This is the second line.
This is the the third line
```

Summary

In this article, we've looked at a number of the most useful string handling commands in PHP. We've seen how to work with strings in different ways and also how to format user data so that it's safe to enter into a database, or display on a web page.

In the final section we looked at formatting raw data in different ways, so that is displayed, correctly formatted, on the page.