# Some Useful String Functions

* empty():Determine whether the $variable is empty.
* is\_null():Determine whether the $variable is NULL. Returns TRUE if it is NULL. Note: An undeclared $variable will return TRUE but may return an error.
* isset(): Accepts multiple $variables separated by commas, but will only return TRUE if all variables are set. Determine whether $variable has been set/created/declared.
* unset() – Destroy/delete a variable or multiple variables
* is\_array(): Determine whether the $variable is an array. Returns TRUE if it is an array. is\_int() : Also known as: is\_integer() determine whether the $variable is an integer. Returns TRUE if it is an integer.
* is\_string():Determine whether the $variable is a string. Returns TRUE if it is a string.
* is\_numeric(): Dtermine whether the $variable is an integer or a numeric string.

**Printing String**

There are four ways to send output to the browser. The **echo** construct lets you print many values at once, while **print( )** prints only one value. The **printf( )** function builds a formatted string by inserting values into a template. The **print\_r( )** function is useful for debugging—it prints the contents of arrays, objects, and other things, in a more-or-less human-readable form.

print**\_r( ) and var\_dump( )**

The print\_r( ) construct intelligently displays what is passed to it, rather than casting everything to a string, as echo and print( ) do. Strings and numbers are simply printed. Arrays appear as parenthesized lists of keys and values, prefaced by Array:

$a = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');

print\_r($a);

**Array**

**(**

**[name] => Fred [age] => 35 [wife] => Wilma**

**)**

Boolean values and NULL are not meaningfully displayed by print\_r( ):

**print\_r(true)**; print 1

**print\_r(false);** does not print anything

**print\_r(null);** also does not print anything.

For this reason, var\_dump( ) is preferable to print\_r( ) for debugging. The var\_dump( )

function displays any PHP value in a human-readable format:

**var\_dump(true);** prints bool(true)

**var\_dump(false);** prints bool(false);

**var\_dump(null);** prints bool(null);

**var\_dump(array('name' => Fred, 'age' => 35));** prints array(2) { ["name"]=> string(4) "Fred" ["age"]=>int(35)}

**strlen( )** : function returns the number of characters in a string:

**EG:**

string = 'Hello, world';

$length = strlen($string); // $length is 12

You can use array syntax on a string, to address individual characters:

**EG:**

$string = 'Hello';

for ($i=0; $i < strlen($string); $i++) {

printf("The %dth character is %s\n", $i, $string[$i]);

}

The 0th character is H

The 1th character is e

The 2th character is l

The 3th character is l

The 4th character is o

Cleaning Strings

Removing Whitespace

You can remove leading or trailing whitespace with the trim( ), ltrim( ), and rtrim( )

functions:

**Syntax:**

$trimmed = trim(string [, charlist ]);

$trimmed = ltrim(string [, charlist ]);

$trimmed = rtrim(string [, charlist ]);

* **trim( )** returns a copy of string with whitespace removed from the beginning and the end.
* **ltrim( )** (the l is for left) does the same, but removes whitespace only from the start of the string.
* **rtrim( )** (the r is for right) removes whitespace only from the end of the string. The optional charlist argument is a string that specifies all the characters to strip.

**For example:**

$title = " Programming PHP \n";

$str\_1 = ltrim($title); // $str\_1 is "Programming PHP \n"

$str\_2 = rtrim($title); // $str\_2 is " Programming PHP"

$str\_3 = trim($title); // $str\_3 is "Programming PHP"

Changing Case

PHP has several functions for changing the case of strings:

**strtolower( )** and **strtoupper( )** operate on entire strings, **ucfirst( )** operates only on the first character of the string, and **ucwords( )** operates on the first character of each word in the string. Each function takes a string to operate on as an argument and returns a copy of that string, appropriately changed.

**For example:**

$string1 = "FRED flintstone";

$string2 = "barney rubble";

print(strtolower($string1));

print(strtoupper($string1));

print(ucfirst($string2));

print(ucwords($string2));

**Output:**

fred flintstone

FRED FLINTSTONE

Barney rubble

Barney Rubble

**Other Functions:**

1. **strlen()** : returns the length of sring

2. **str\_word\_count()** : returns the number of words in string

3**. strrev()** : reverse the string

4**. strops()** : finds the specific text within a string

5**. str\_replace()** : replace some characters with some other characters in string

6. **addslashes():** the addslashes() function returns a string with backslashes in front of predefined characters.The predefined characters are:

* single quote (')
* double quote (")
* backslash (\)
* NULL

**Example:**

$str = "Who's Peter Griffin?";

echo addslashes($str); // output will be : Who\'s Peter Griffin?

7. **chunk\_split()** : function splits a string into a series of smaller parts. **Note:** This function does not alter the original string.

**Example:**

$str = "Hello world!";  
echo chunk\_split($str,1,"."); // output : H.e.l.l.o. .w.o.r.l.d.!.

8. **explode() :** function breaks a string into an array. **Note:** The "separator" parameter cannot be an empty string.

**Syntax:**

explode(*separator,string);*

Example:

$str = 'one,two,three,four';

$arr= explode(',',$str,0);

print\_r($arr); // output will be : Array ( [0] => one [1] => two [2] => three [3] => four )

9. **implode()** function returns a string from the elements of an array. **Note:** The implode() function accept its parameters in either order. However, for consistency with explode(), you should use the documented order of arguments. **Note:** The separator parameter of implode() is optional. However, it is recommended to always use two parameters for backwards compatibility.

Syntax:

implode(*separator,array*)

Example:

$arr = array('Hello','World!','Beautiful','Day!');

echo implode(" ",$arr)."<br>";

echo implode("+",$arr)."<br>";

echo implode("-",$arr)."<br>";

echo implode("X",$arr);

Output:

Hello World! Beautiful Day!

Hello+World!+Beautiful+Day!

Hello-World!-Beautiful-Day!

HelloXWorld!XBeautifulXDay!

10. number\_format() function formats a number with grouped thousands.

###### Example:

echo number\_format("1000000")."<br>";

echo number\_format("1000000",2)."<br>";

echo number\_format("1000000",2,",",".");

###### Output:

1,000,000  
1,000,000.00  
1.000.000,00

11. printf() function outputs a formatted string.Possible format values:

%% - Returns a percent sign

%b - Binary number

%c - The character according to the ASCII value

%d - Signed decimal number (negative, zero or positive)

%e - Scientific notation using a lowercase (e.g. 1.2e+2)

%E - Scientific notation using a uppercase (e.g. 1.2E+2)

%u - Unsigned decimal number (equal to or greather than zero)

%f - Floating-point number (local settings aware)

%F - Floating-point number (not local settings aware)

%g - shorter of %e and %f

%G - shorter of %E and %f

%o - Octal number

%s - String

%x - Hexadecimal number (lowercase letters)

%X - Hexadecimal number (uppercase letters)

**Example:**

$number = 9;  
$str = "Beijing";  
printf("There are %u million bicycles in %s.",$number,$str);

**Output**

There are 9 million bicycles in Beijing.

# PHP include and require Statements

The include or require statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

**The include and require statements are identical, except upon failure:**

* require will produce a fatal error (E\_COMPILE\_ERROR) and stop the script
* include will only produce a warning (E\_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use the **include** statement otherwise use **require**

### Syntax

### include '*filename*'; or require '*filename*';

### Example:

<html>  
<body>  
<h1>Welcome to my home page!</h1>  
<p>Some text.</p>  
<p>Some more text.</p>  
<?php include 'footer.php';?> // or <?php require 'footer.php';?>   
</body>  
</html>

And footer.php is

<?php

echo 'Copyright © 1999-2018 W3Schools.com';

?>

# require\_once

The require\_once statement is identical to [require](http://php.net/manual/en/function.require.php) except PHP will check if the file has already been included, and if so, not include (require) it again.