

PHP User Defined Functions

Besides the built-in PHP functions, we can create our own functions.

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads.

A function will be executed by a call to the function.

Create a User Defined Function in PHP

A user defined function declaration starts with the word "function":

Syntax

```
function functionName() {  
    code to be executed;  
}
```

Note: A function name can start with a letter or underscore (not a number).

Tip: Give the function a name that reflects what the function does!

Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // call the function
?>
```

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (\$fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

The following example has a function with two arguments (\$fname and \$year):

Example

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}
familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

PHP Functions - Returning values

To let a function return a value, use the return statement:

PHP 5 Arrays

An array stores multiple values in one single variable:

Example

```
<?php
$scars = array("Volvo", "BMW", "Toyota");
echo "I like " . $scars[0] . ", " . $scars[1] . " and " . $scars[2] . ".";
?>
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single

```
$scars1 = "Volvo";
$scars2 = "BMW";
$scars3 = "Toyota";
```

variables could look like this:

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys

- **Multidimensional arrays** - Arrays containing one or more arrays
-

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
```

```
$cars[1] = "BMW";
```

```
$cars[2] = "Toyota";
```

The following example creates an indexed array named \$cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Get The Length of an Array - The count() Function

The count() function is used to return the length (the number of elements) of an array:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
```

Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a for loop, like this:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);
for($x = 0; $x < $arrlength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
?>
```

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them. There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- sort() - sort arrays in ascending order
 - rsort() - sort arrays in descending order
 - asort() - sort associative arrays in ascending order, according to the value
 - ksort() - sort associative arrays in ascending order, according to the key
 - arsort() - sort associative arrays in descending order, according to the value
 - krsort() - sort associative arrays in descending order, according to the key
-

Sort Array in Ascending Order - sort()

The following example sorts the elements of the \$cars array in ascending alphabetical order:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
?>
```

The following example sorts the elements of the \$numbers array in ascending numerical order:

Example

```
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
?>
```

Sort Array in Descending Order - rsort()

The following example sorts the elements of the \$cars array in descending alphabetical order:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);
?>
```

The following example sorts the elements of the \$numbers array in descending numerical order:

Example

```
<?php
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers);
?>
```

Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
?>
```

Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
?>
```

Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

```
<?php
$page = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($page);
?>
```