A sample Training Module from our course
WELL HOUSE CONSULTANTS LTD
404, The Spa • Melksham, Wiltshire SN12 6QL • United Kingdom
PHONE: 01225 708225 • FACSIMLE 01225 707126 • EMAIL: info@wellho.net

174

Functions

If you have a piece of code that you want to share between a number of PHP web pages, write it into a function! A function is a named block of code into which you can pass a number of variables (known as parameters) and from which a result can be returned.

If you find yourself repeating a block of code, it might be that you should be using a loop. But a loop isn't always appropriate, for example if you have an operation that you want to perform at different places in the same program, or even in different programs.

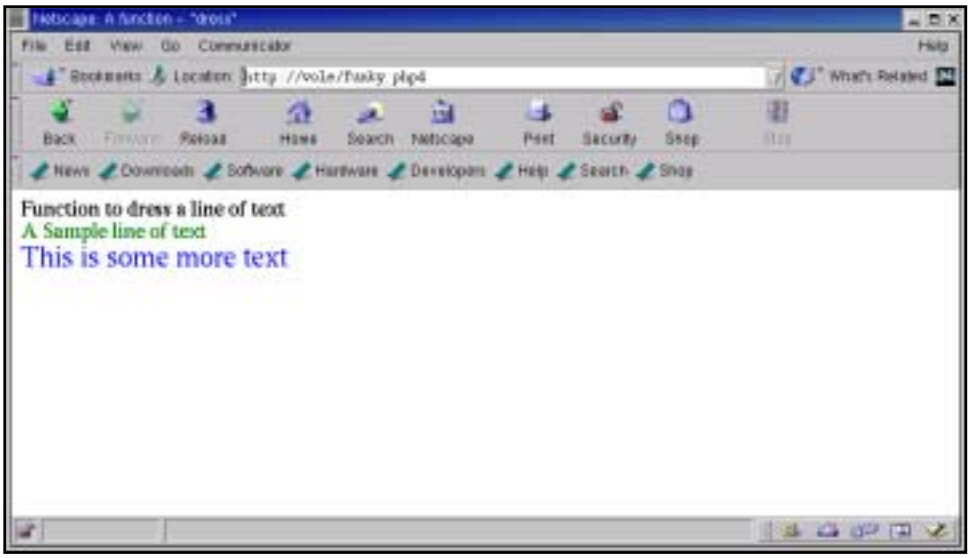
Blocks that you wish to repeat in this way are best written as functions. A function is a piece of code that can be run from elsewhere, passing into it any changeable values, and it will pass back its answer.

As well as saving the need to repeat code, functions divide medium-to-large sized programs into a series of much more manageable blocks, making coding, testing and debugging much easier. Each function has an explicitly designed interface, so that the author can write, test and debug it in relative isolation. And each function has its own set of variables so that you don't get problems when two authors both happen to use a variable called (for example) "$result".

```
<head>
<title>A function - "dress"</title>
</head>
<body bgcolor=white>
Function to dress a line of text<br>
<?php
function dress($text,$colour,$size) {
               $tagged = "<font color=$colour size=$size>$text</
font><br>";
               return $tagged;
}
################################################################

$textline = "A Sample line of text";
$also = "This is some more text";

$tagged = dress($textline,"green",3);
$t2 = dress($also,"blue",6);

print ("$tagged$t2");

?>
</body>
```

In this example, we define a function called "dress" that has three parameters. Within the function we refer to the parameters as "$text", "$colour" and "$size". The function adds font tags onto the start and end of our text, and also a line break which could be something we commonly want to do through a whole big piece of PHP code. The function is defined as returning the contents of its (the function's) **`$tagged`** variable.

Figure 814 Running "funky.php4"



The function is not executed at the point it's defined, it is just defined, and it's up to us to actually call in that code later in our PHP program. In this example program, we've done so twice, once with the text in the **`$textline`** variable, and another time with the text in the **`$also`** variable. We're allowed to call functions with variable and constant parameters,[1] but the order of the parameters is important.

In our example, we've actually got two variables, both called "$tagged" to illustrate PHP's default scoping. In practice you might look to avoid too much of this as it's confusing.

174.1 Global and static variables

By default, variables in a function are created each time the function is run, are not shared with the main program, and thus are local. There may be occasions that you want to change this behaviour:

1.If you declare a variable in a function to be **`global`**, it will be the same variable that is used in your main program

2.If you declare a variable to be **`static`**, it will not be the same as any variable of the same name in your calling code, but it will be retained between calls.

[1] we do both here

Lets's add three "counters" to our last example to see how each variable type works:

Figure 8.15 Adding three counters to our example.

```
<head>
<title>A function - "dress"</title>
</head>
<body bgcolor=white>
Function to dress a line of text<br>
<?php
function dress($text,$colour,$size) {
  global $c1;
  static $c2 = 1;
  $c3 = 1;

  $c1++; $c2++; $c3++;

  $tagged = "<font color=$colour size=$size>$text
             global: $c1
             static: $c2
             local: $c3</font><br>";
  return $tagged;
}
################################################################

$c1 = 15;
$c2 = 20;
$c3 = 25;

$textline = "A Sample line of text";
$also = "This is some more text";

$tagged = dress($textline,"green",3);
$t2 = dress($also,"blue",6);

print ("$tagged$t2");

print ("Main values $c1 $c2 and $c3");
?>
</body>
```

174.2 Loading functions from another file

You can load functions into your PHP page from another file using the **include, include_once**, **require** or **require_once** statement. We strongly suggest that you put any such calls outside program loops if you can. The difference is that a **require** will produce a fatal error if the required file is not available, but **include** will only produce a warning. The **_once** form of each of these functions will only be performed once each time you run your PHP, but the regular form will repeatedly drag in the same code if it's in a loop.

On a training course and during initial development, you'll probably want to keep your files of functions in the same directory as your main PHP script, just to save the hassle of managing lots of different directories. Once you're going to place code on a live site, you should move your included and required files to a separate directory that's not accessible through its own URL.[1]

174.3 Defaulting parameters

If you call a function with fewer parameters than it's formally declared with, you can default the unpassed parameters within the function by assigning them values in the declaration.

174.4 Call by value v call by name

Functions in PHP are usually "called by value". In other words, the value that you call a function with is copied into a separate variable within the function. This default behaviour means that functions won't unwittingly change the contents of a variable within the code that calls the function.

It's usually regarded as poor programming practice when writing functions to change the values of incoming variables in any case, but there are times that you might want to do it. In PHP, you can specify that a parameter is to be passed "by name" rather than "by value" by adding an **&** in front of the variable name in the declaration of the function.

If you're familiar with subroutines and functions in Perl or C, you will recall they default to calling by value. You can "call by name" using a **** in Perl or an ***** in C in the call, unlike in PHP, where it's only the function that changes. In Java, objects are always passed by name anyway.

Example

In this example, we'll load a function from another file and call it with a variable number of arguments, one of which we'll call by name (or reference) rather than by value.

The main php program (*funk3.php4*):

```
<head>
<title>Function(s) in a separate file</title>
</head>
<body bgcolor=white>
Function to dress a line of text<BR>
<?php
include ("dressing.inc");

$textline = "A Sample line of text";
$also = "This is some more text";
$final = "Some standard issue text";
```

[1] You'll learn how to do this at the same time that you learn about special server variables and settings.

```
dressing($textline,"green",3);
dressing($also,"blue");
dressing($final);

print ("$textline$also$final");

?>
</body>
```

The included file (*dressing.inc*):

```
<?php
# Sample include file; also defaults colour and size
function dressing(&$text,$colour = "fushia",$size = 8) {
  $text = "<font color=$colour size=$size>$text</font><br>";
}
?>
```

If you're writing a file from which you'll be including just one function, it's good practice to give the function and the file the same name (and not to use a php4 extension so that no one tries to run the included file stand-alone). If there are several functions in a file, you should look to come up with an appropriate collective name for them for the file and included name.

The output from running the program::

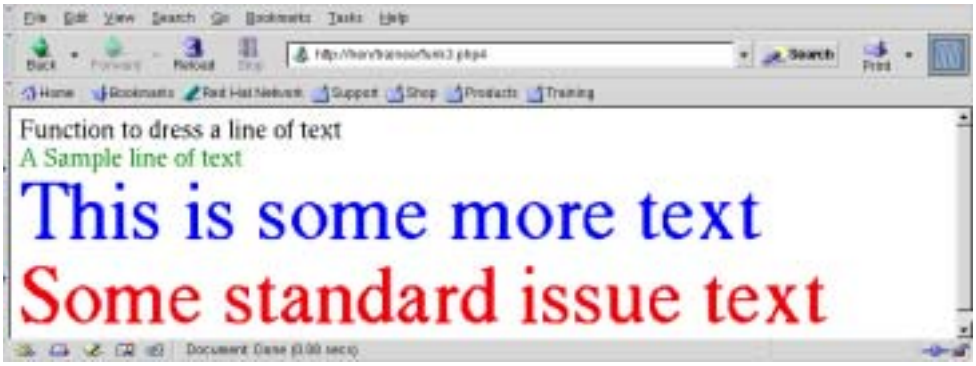


Figure 816 Running the program.

174.5 Object oriented PHP

As your PHP application grows from tiny to small, you'll want to start using functions to reduce the size of code chunks into manageable, named blocks that are independently testable and reusable. As it grows beyond a small application, you'll want to split off those functions into separate include files so they can be shared between different pages without code duplication, and they can be kept in logical and easier-to-manage groups.

This model starts to break down as your application moves in size from medium to large. For example, if you're loading include files from several different sources, chances are that there might be two functions with the same name, because they were written by two independent programmers. It doesn't have to be *your* code that provides such a conflict; it could be two modules loaded from the Pear (PHP extension and Application Repository), or one of their codes and one of yours.

If your application is going to be a large one, you should consider using object oriented design. With object orientation, you start by considering the data types you'll be using and all the various methods you need to access them. Object orientation is very much a way of thinking about your design rather than a programming language as such, but most modern languages (including PHP) provide excellent facil-

ities to make it easy to achieve an OO-based suite of applications.

Functions are a necessity for most PHP programmers, but object orientation doesn't make sense for everyone, in just the same way that you probably wouldn't think it was sensible to take an airline flight from London to Birmingham, but it would be an option from London to Aberdeen and probably the first choice from London to Toronto.

PHP's roots are in smaller applications, but PHP 4 does include an object model and that's been enhanced to make it an excellent model in PHP5.

Exercise