Never trust a computer you can't throw out a window.

Steve Wozniak

5 Functions and includes

5.1 Introduction

This chapter is about the reuse of pieces of code. When you have a piece of logic that you want to use more than once, it probably is a good idea to make a function for it. Functions are the easiest way to write reusable code and therefore are good practise. Don't just copy code to reproduce a certain logic. Instead write a function so you can easely reuse this piece of logic.

And when you have some piece(s) of code that you want to use in more then one PHP page, put them in one or more different files and include these files when needed.

5.2 Declaring a function

Functions are declared using the `function()` statement. They are case insensitive.

You can add parameters. If you want a parameter to be optional, supply a default value. But you can't declare a mandatory parameter after an optional one.

A function can return a value. This is done by using the `return` statement. Calling a return statement also breaks the execution of the function.

Listing 5.1 Declaration of a function

```
1  <?php
2  // A function that returns the get-value for a given key.
3  // If the key doesn't exists, it returns an optional default-value.
4
5  function giveGetValue($key, $default = false)
6  {
7          if (isset($_GET[$key])) {
8                  return $_GET[$key];
9          } else {
10                 return $default;
11         }
12 }
13 ?>
```

An simple example can be seen at `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex5.1.php` (Remember that you can put an 's' after .php to view the php-source code).

There are two functions defined: one for the header and one for the footer. By reusing this function, a lot of work is saved as we don't have to copy the headers and footers all over again. Moreover: if we want to change something that should be visible for all pages of our website, we only have to change it once.

In a next section we will see how you can include one PHP file into another.

5.3 Functions and their memory space

Functions have their own memory space, which means they use their own set of variables. E.g. is you have declared a variable `$somevar` outside a function, this variable won't exist in the memoryspace of the function and cannot be used inside this function.

Listing 5.2 functions have their own memory space

```
<?php
// declaration of a variable
$somevar = 'blabla';

function doSomethingWonderful()
{
        // the next line will trigger a notice and won't print a thing
        // because the var $somevar doesn't exists within the funcion's memory
        echo $somevar;
}
?>
```

There are two main ways to overcome this. The first one is by passing the variable as a parameter (see the previous section for an example).

The second way is by making the variable global and using this globalised variable inside the function scope. To do this, you must 1) declare the variable outside the function and before you declare the function and 2) globalise the variable inside the function.

Listing 5.3 making a variable global

```
<?php

$somevar = 'blabla';

function doSomethingWonderful()
{
        // pointing to the global var
        global $somevar;

        // this now will print 'blabla'
        echo $somevar;
}
?>
```

5.4 Difference between passing by value and global

There is a big difference between the two ways to pass variables to a function.

If you pass the variable as a parameter, the current value is copied to the variable inside the function. There is no connection between the two. If you use the variable inside the function as a global one, there is only one variable. See the example below.

Listing 5.4 difference between global and parameter

```php
<?php

$somevar = 'blabla';

function doSomethingWonderful($somevar)
{
        // $somevar contains a copy of the value of the
        // variable passed to this function as a parameter

        $somevar .= ' is what he says';

        echo $somevar;
}

function doSomethingWonderfulGlobal()
{
        global $somevar;

        // $somevar is a pointer to $somevar
        // it is the same variable

        $somevar .= ' is what he says';

        echo $somevar;
}

// parameter use
doSomethingWonderful($somevar); // will print 'blabla is what he says'

echo $somevar; // will print 'blabla'

// global use
doSomethingWonderfulGlobal(); // will print 'blabla is what he says'

echo $somevar; // will also print 'blabla is what he says'

?>
```

You can also pass a variable by reference, but from PHP version 5.3 this method is depreciated. More info at `www.php.net/manual/en/language.references.pass.php`.

5.5 Form validation

When process data a user sends through a from, any form of validation is required. e.g. has the user entered a valid birthday or is the username already taken or not?

Listing 5.5 form to be sent and validated

```
<form action="https://dynweb.webontwerp.khleuven.be/post.php" method="post">
        <div class="control-group">
            <label for="firstname">Firstname</label>
            <input id="firstname" name="firstname" type="text" />
        </div>
    <div class="form-actions">
        <input type="hidden" name="ahiddenfield" value="ahiddenvalue" />
        <button type="submit" class="btn btn-primary">submit</button>
    </div>
</form>
```

There are two fields in the example above: a field with name 'firstname' and a field with name 'ahiddenfield'. The form uses a `POST` method to submit the data, based on the attribute `method="post"`. The URI to which the POST request is performed (the script that is going to be used to process the form data), is defined in the `action` field. (More on `$_GET` and `$_POST` in later chapters)

Listing 5.6 php code to process and validate the formdata, without the use of functions

```
<?php
// without a function
if ((isset($_POST['firstname'])) && (trim($_POST['firstname']))) {
        $firstname = trim($_POST['firstname']);
} else {
        $firstname = false;
}

if ((isset($_POST['ahiddenfield'])) && (trim($_POST['ahiddenfield']))) {
        $hiddenvalue = trim($_POST['ahiddenfield']);
} else {
        $hiddenvalue = false;
}
?>
```

Needless to say that this copy- and pasting needs te be refactored by the use of a function:

Listing 5.7 php code to process and validate the formdata, without the use of functions

```
<?php
function getNotEmptyPostValue($key)
{
        if ((isset($_POST[$key])) && (trim($_POST[$key]))) {
                return trim($_POST[$key]);
        } else {
                return false;
        }
}

$firstname = getNotEmptyPostValue('firstname');
$hiddenvalue = getNotEmptyPostValue('hiddenvalue');
?>
```

5.6 Includes

A common thing you want to do is to split up your project into different files. In stead of copying some functions you want to use throughout your project, put them in a seperate file and include this file on every page you want to use these functions.

By default, if a request for a PHP file is made to the server. Only that PHP file is loaded. Functions, constants, ... declared in other files are therefore not accessible.

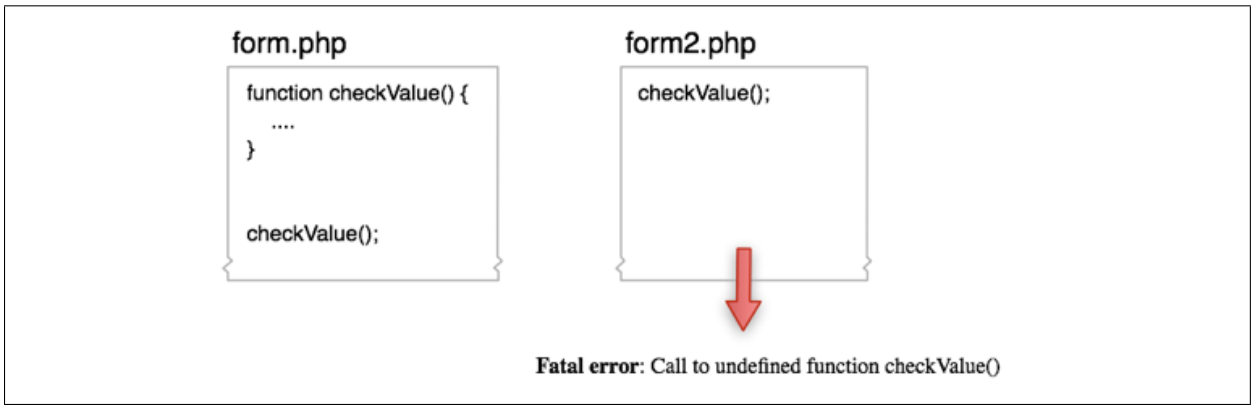

Fig. 5.1 checkValue() doesn't exist when form2.php is executed

There are four functions to include a piece of code into another page: `include()`, `require()`, `include_once()`, `require_once()`.

The two functions `include()` and `require()` are identical in every way, except how they handle errors when the included file isn't found:

- `include()` generates a warning, but the script will continue execution. This can be very confusing.
- `require()` generates a fatal error, and the script will stop

The two functions `include_once()` and `require_once()` differ from the other two in the way that they keep a memory of the files already included and won't include the same file twice.

5.7 Includes: good practise

So the function to use that shows the most debug info and by that is the least error-prone is `require_once()`.

When you use one of these, all of the contents of the included file is merged into the document that includes it, at the linenumber where you called the include function. So if you make some echo's or called some functions, these too will be included.

This probably isn't what you want either.

It is best practise to seperate your logic. Seperate your database transactions, seperate your controller logic and seperate your presentation logic. Think generic. Don't just copy.

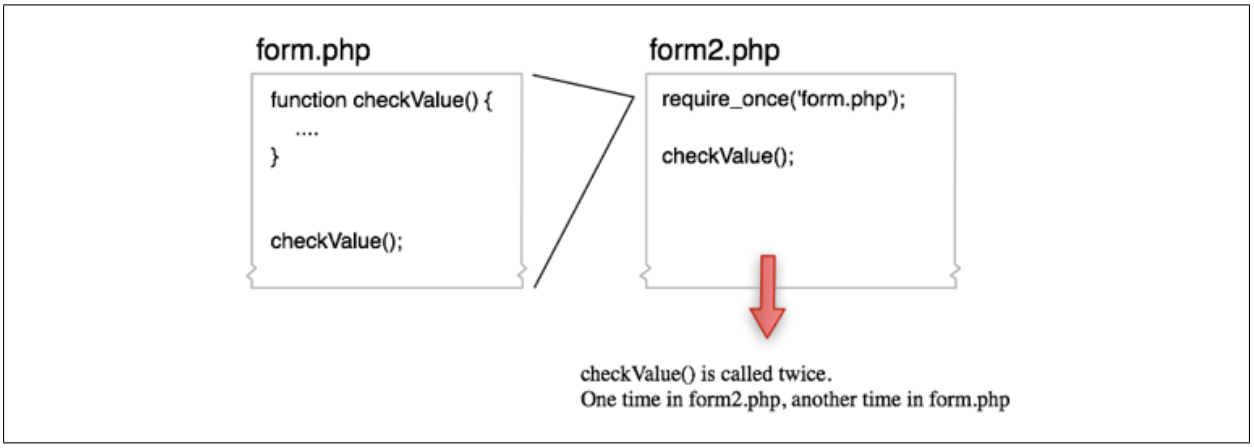

Fig. 5.2 form.php is completely executed inside form2.php

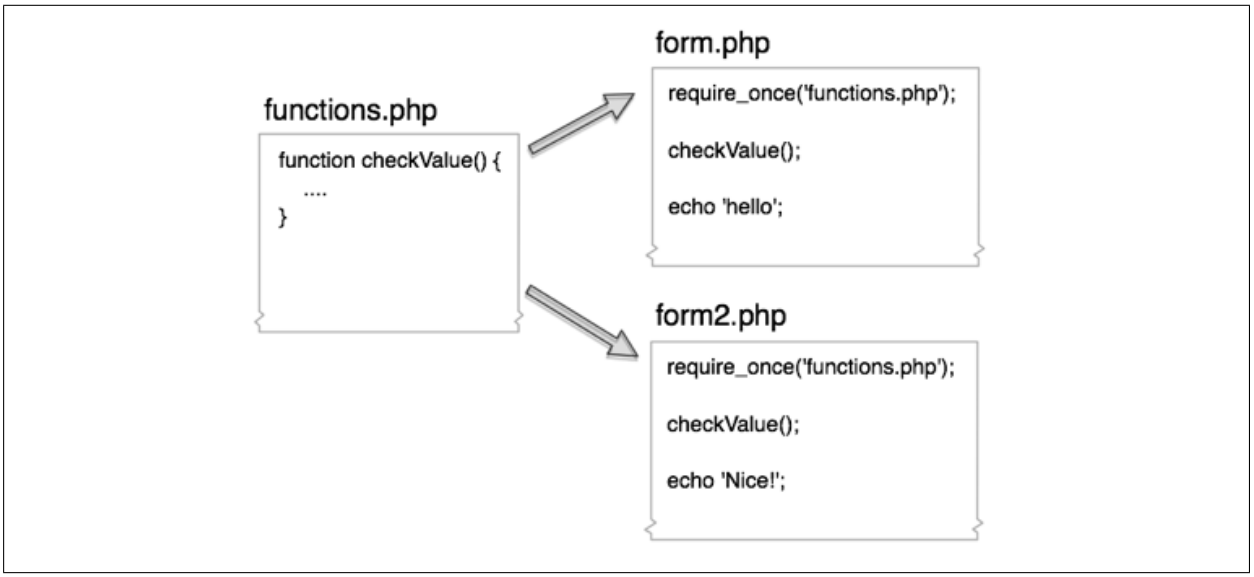

Fig. 5.3 Put functions in separate files from executed pages.

Have a look at the example at `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex5.2.php`, also have a look at the PHP source-code from `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex5.2.includes.phps`.