COMP519 Practical 13
PHP (3)

Introduction

- This practical is dedicated to PHP Programming. While you work through the exercises below compare your results with those of your fellow students and ask for help and comments if required.
- You might proceed more quickly if you cut-and-paste code from this PDF file. Note that a cut-and-paste operation may introduce extra spaces into your code. It is important that those are removed and that your code exactly matches that shown in this worksheet.
- The exercises and instructions in this worksheet assume that you use the Department's Linux systems to experiment with PHP.
- To keep things simple, we will just use a text editor, a terminal, and a web browser. You can use whatever text editor and web browser you are most familiar or comfortable with.
- If you do not manage to get through all the exercises during this practical session, please complete them in your own time.

Exercises

1. In the lectures we covered the way in functions are defined and called in PHP as well as the concept of local and global variables. In this exercise, we want to study the *scope of variables* in more detail.
   a. Create a file name `php13A.php` in `$HOME/public_html` with the following HTML markup and PHP code. Make sure that the file permissions of that file are correct.

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
    <title>PHP 13A</title>
  </head>
  <body>
    <h1>Variable Scope</h1>
<?php
  error_reporting( E_ALL );
  ini_set('display_errors', 1);
  ini_set('display_startup_errors', 1);
  echo "<h2>Exercise 1</h2>\n";
  // This is a global variable
  $a = 0;

  if ($a == 0) {
     // This is still a global variable
     $b = 1;
  }

 function checkVar($context,$name,&$var) {
```

```
    if (isset($var)) {
       echo "$context $name=$var<br>\n";
    } else {
       echo "$context $name is not defined<br>\n";
    }
 }

 function func1($c) {
    # $c and $d are local variables
    for ($d = 0; $d <= 5; $d++) {
    }
    checkVar("In func1",'$a',$a);
    checkVar("In func1",'$b',$b);
    checkVar("In func1",'$c',$c);
    checkVar("In func1",'$d',$d);
 }
 func1(3);

 checkVar("Outside func1",'$a',$a);
 checkVar("Outside func1",'$b',$b);
 checkVar("Outside func1",'$c',$c);
 checkVar("Outside func1",'$d',$d);
?>
 </body>
</html>
```

b. Open a web browser and access the URL

   `https://student.csc.liv.ac.uk/~<user>/php13A.php`

   were *<user>* should be replaced by your user name. Besides headings the page should contain the following:

```
In func1 $a is not defined
In func1 $b is not defined
In func1 $c=3
In func1 $d=6
Outside func1 $a=0
Outside func1 $b=1
Outside func1 $c is not defined
Outside func1 $d is not defined
```

   What is the output telling you about the scope of the variables `$a`, `$b`, `$c`, and `$d`?

c. Replace the line

```
$a = 0;
```

   by

```
$a = 1;
```

   Save the changes, then reload the page. How has the output changed and why?

d. In the definition of the function `func1`, add the line

```
$a = 2;
```

before the for-loop in that function. Save the changes, then reload the page. How has the output changed and why?

e. In the definition of the function `func1`, add the line

```
global $a;
```

before the assignment `$a = 2` that you have introduced in the previous step. Save the changes, then reload the page. How has the output changed and why?

f. Finally, what happens if you move the line

```
global $a;
```

after the assignment `$a = 2` in `func1`? Give it a try. What does that tell you about where declarations like this need to be placed?

2. Let us move on to nested functions and their effect on the scope of variables.

a. Create a file named `php13B.php` in `$HOME/public_html` with the following HTML markup and PHP code. Make sure that the file permissions of that file are correct.

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
    <title>PHP 13B</title>
  </head>
  <body>
    <h1>Nested Functions and Variable Scope</h1>
<?php
error_reporting( E_ALL );
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
echo "<h2>Exercise 2a</h2>\n";
function outside1() {
   $msg = "Outside!";
   function inside1() {
      $msg = "Inside!";
      echo "(I1): $msg<br>\n";
   }
   inside1();
   echo "(O1): $msg<br>\n";
}
outside1();
?>
  </body>
</html>
```

b. Open a web browser and access the URL

`https://student.csc.liv.ac.uk/~`*`<user>`*`/php13B.php`

were *`<user>`* should be replaced by your user name. Besides headings the page should contain the following:

```
(I1): Inside!
(O1): Outside!
```

Based on Exercise 1 we know why this is so: `$msg` in `inside1()` has nothing to do with `$msg` in `outside1()`.

c. Would a `global` declaration connect the two different instances of `$msg`? Add the declaration

```
global $msg;
```

as the first statement of the function `inside1()`. Save your changes then reload the web page. The output does not change, why not?

d. There is no `nonlocal` declaration in PHP, so we have to try something different. Add the following PHP code just after the call of `outside1()`:

```
echo "<h2>Exercise 2d</h2>\n";
function outside2() {
   $msg = "Outside!";
   function inside2() {
      $msg = "Inside!";
      echo "(I2): $msg<br>\n";
   }
   $inside3 = function () use($msg) {
      echo "(I3a): $msg<br>\n";
      $msg = "Inside!";
      echo "(I3b): $msg<br>\n";
   };
   inside2();
   $inside3();
   echo "(O2): $msg<br>\n";
}
outside2();
```

Save the file and reload the page in your web browser. You should the following additional output:

```
(I2): Inside!
(I3a): Outside!
(I3b): Inside!
(O2): Outside!
```

From the output we can see that `$msg` inside `$inside3()` now inherits its value from `outside2()`, but changing it has not effect on the value of `$msg` in `outside2()`. It is as if `$inside3()` has a parameter and was passed `$msg` has an argument.

e. Change the line

```
$inside3 = function () use($msg) {
```

to

```
$inside3 = function () use(&$msg) {
```

Save the file and reload the page in your web browser. You should see that `$inside3()` is now able to change the value of `$msg`, as one would expect if an argument is passed by reference.

f. But where exactly does `$inside3()` take `$msg` from? Is it determined at the time we assign the anonymous function to `$inside3()` or at the time we call that function?

Add the line

```
$msg = "Still outside!"
```

before the call of `$inside3()`. Save the file and reload the page in your web browser. What does the output tell you about the answer to the question above?

g. Add a second call to `outside2()` at the end of the script. Save the file then reload the page in your web browser. You should see an error message. What is going wrong? Try to change the code for `outside2()` so that a second call is possible?

h. Let us see whether array variables behave differently to scalar variables in PHP. Add the following code at the end of the script:

```
echo "<h2>Exercise 2h<h2>\n";
function outside4() {
   $arr[0] = "Outside!";
   function inside4() {
      $arr[0] = "Inside!";
      echo "(I4): ",$arr[0],"<br>\n";
   }
   inside4();
   echo "(O4): ",$arr[0],"<br>\n";
}
outside4();
```

Save the file and reload the page in the web browser. What does the output tell you about PHP arrays?

3. In this exercise we want to explore *variable-length argument lists* and *keyword arguments*.

a. Create a new HTML document `php13C.php` in your `public_html` document with the following content:

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
    <title>PHP 13C</title>
    <META name="description" content="php13C.php">
  </head>
  <body>
  <h1>Variable-length Argument Lists</h1>
<?php
try {
   echo "1: ", reduceOp(2,3), "<br>\n";            # throws an exception
} catch (Exception $e) {
   echo "1: Exception ",$e->getMessage(),"<br>\n"; # 'TypeError'
}
try {
   echo "2: ",reduceOp(2,3,array('op' => '/')),    # throws an exception
   "<br>\n";
} catch (Exception $e) {
   echo "2: Exception ",$e->getMessage(),"<br>\n"; # 'ValueError'
}
echo "3: ",reduceOp(array('op'=>'+')),            # should return NULL
     "<br>\n";
echo "4: ",reduceOp(2,array('op' => '*')),        # should return 2
```

```
     "<br>\n";
echo "5: ",reduceOp(2,3,5,array('op' => '+')),     # should return 10
     "<br>\n";
echo "6: ",reduceOp(2,3,5,7,array('op' => '*')),   # should return 210
     "<br>\n";
echo "7: ",reduceOp(2,3,5,7,11,array('op' => '-')),# should return -24
     "<br>\n";
?>
  </body>
</html>
```

Exceptionally, there are comments after the code. Do not do that in code that you develop for the assignments.

b. Extend `php13C.php` with the definition of a function `reduceOp` that behaves as follows:

- `reduceOp` expects an associative array as last argument that maps the key `'op'` to one of the values `'+'`, `'-'`, or `'*'`. In the absence of this argument, an exception with value `'TypeError'` should be raised. If the associative array is present, but does not have a valid value, then an exception with value `'ValueError'` should be raised.
- In addition to this associate array, `reduceOp` takes a possibly empty list of numbers as argument. If a valid associative array is supplied, but the list of numbers is empty, then the value `NULL` should be returned. If an associative array mapping `'op'` to value 'op' as well as a non-empty list $n_1, \ldots, n_k$ of numbers is supplied, then the value $(((n_1\, op\, n_2)\, op\, n_3) \cdots op\, n_k)$ should be returned.
- Check that your `reduceOp` function behaves as expected using the test cases provided.

4. The following exercise deals with the *modification of array elements*, *string operations* and *pattern search and replace*.

a. Create a file name `php13D.php` in `$HOME/public_html` with the following HTML markup and PHP code. Make sure that the file permissions of that file are correct.

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
    <title>PHP 13D</title>
  </head>
  <body>
    <h1>Regular Expressions</h1>
<?php
echo "<h2>Exercise 4</h2>\n";
$names = ["Dave Shield", "Mr Andy Roxburgh",
          "Dr George Christodoulou",
          "Dr Ullrich Hustadt", "Prof Tara Wang"];
foreach ($names as $name)
   echo "(1) Name: $name<br>\n";
// Your own code here
foreach ($names as $name)
   echo "(2) Name: $name<br>\n";
?>
  </body>
</html>
```

Load the page in your web browser. The output produced by the code above should be as follows:

```
(1) Name: Dave Shield
(1) Name: Mr Andy Roxburgh
(1) Name: Dr George Christodoulou
(1) Name: Dr Ullrich Hustadt
(1) Name: Prof Tara Wang
```

and the same again with `(1)` replaced by `(2)`.

b. Add code at the point indicated in the code in Exercise 4a that modifies the strings stored in `$names` so that the output produced will change to:

```
(2) Name: SHIELD, Dave
(2) Name: ROXBURGH, Andy
(2) Name: CHRISTODOULOU, George
(2) Name: HUSTADT, Ullrich
(2) Name: WANG, Tara
```

This should be done with the help of string manipulation operations, see, for example,

http://www.php.net/manual/en/function.preg-match.php
http://www.php.net/manual/en/function.preg-replace-callback.php

and the code should be independent of the actual names stored in `$names`.

Also recall from the lecture notes the variation of a `foreach`-loop that allows you to modify the elements of an array.

c. Add the following code at the end of the script:

```
foreach ($names as $name)
   if (TRUE) {
      echo "(3) Name: $name is valid<br>\n";
   } else {
      echo "(3) Name: $name is invalid<br>\n";
   }
```

d. Replace the boolean constant `TRUE` in the conditional statement with a boolean expression involving one or more calls of the `preg_match` function with regular expressions that evaluates to `TRUE` if and only if `$name` satisfies each of the following conditions:

- there is at most one occurrence of a comma followed by a space;
- in front of the comma is a non-empty sequence of letters A–N and P–Z;
- following the space is a non-empty sequence of letters with one letter occurring at least twice.

If programmed correctly, the output should be

```
(3) Name: SHIELD, Dave is invalid
(3) Name: ROXBURGH, Andy is invalid
(3) Name: CHRISTODOULOU, George is invalid
(3) Name: HUSTADT, Ullrich is valid
(3) Name: WANG, Tara is valid
```