COMP284 Practical 3
PHP (2)

Introduction

- This worksheet contains exercises that are intended to familiarise you with 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 the 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 before the next practical takes place.

Exercises

1. Our first exercise is about *arrays* and *array operators* in PHP.

   a. In the lectures we have seen that PHP has array operators that allow us to use arrays as stacks or queues.

   b. Use a text editor to create a file `php03A.php` in your `public_html` directory with the following content:

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
    <title>PHP 03A</title>
  </head>
  <body>
    <h1>Array Operators</h1>
<?php
echo "<h2>Exercise 1a</h2>\n";
$planets = array("earth");
array_unshift($planets,"mercury","venus");
array_push($planets,"mars","jupiter","saturn");
echo "(1) \$planets = [",join(", ",$planets),"]<br>\n";
$last = array_pop($planets);
echo "(2) \$planets = [",join(", ",$planets),"]<br>\n";
$first = array_shift($planets);
echo "(3) \$planets = [",join(", ",$planets),"]<br>\n";
```

```
echo "(4) \$first = $first, \$last = $last<br>\n";
?>
  </body>
</html>
```

Make sure that the file permission of the file are correct.

c. Open a web browser and access the URL

https://student.csc.liv.ac.uk/~*<user>*/php03A.php

were *<user>* should be replaced by your user name.

You should see the output produced by the PHP script showing you how the various array operators change an array. Make sure that you understand how each operator works.

d. Assignments involving arrays work differently in PHP than in Java and Python. To verify that add the following code at the end of the PHP script in `php03A.php`:

```
echo "<h2>Exercise 1c</h2>\n";
$spheres = $planets;
echo "(5) \$spheres = [",join(", ",$spheres),"]<br>\n";
$planets[1] = "midgard";
echo "(6) \$planets = [",join(", ",$planets),"]<br>\n";
echo "(6) \$spheres = [",join(", ",$spheres),"]<br>\n";
$spheres = &$planets;
echo "(7) \$spheres = [",join(", ",$spheres),"]<br>\n";
$planets[0] = "friga";
echo "(8) \$planets = [",join(", ",$planets),"]<br>\n";
echo "(8) \$spheres = [",join(", ",$spheres),"]<br>\n";
```

Save the file and reload the papge in your web browser. What does the output tell us about how array assignments work in PHP?

e. Extend the script with code that repeatedly removes elements from the front of the array `$planets` until the array is empty and in each step prints out the element that was removed as well as the elements that remain in the array. The output should look as follows:

```
Removed: friga Remaining: [midgard, mars, jupiter]
Removed: midgard Remaining: [mars, jupiter]
Removed: mars Remaining: [jupiter]
Removed: jupiter Remaining: []
```

2. In PHP all arrays are *associative arrays*.

a. Create a new PHP script called `php03B.php` in your `public_html` directory with the following content

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
    <title>PHP 03B</title>
  </head>
  <body>
    <h1>Associative Arrays</h1>
```

```
<?php
$dict1 = array('a' => 1, 'b' => 4);
$dict2 = $dict1;
echo "\$dict1['a'] = ",$dict1['a'],"<br>\n";
echo "\$dict2['b'] = ",$dict2['b'],"<br>\n";
?>
  </body>
</html>
```

Open the web page in your web browser to make sure that it works.

b. Add code for a loop that prints out all the key-value pairs in `$dict1`.

c. A possible application of an associative array is to count the number of occurrences of particular strings. For example, `$dict1` may be seen to store the information that the character 'a' has one occurrence and the character 'b' four occurrences in some text. We want to develop code that performs such a count.

Add the following code at the end of your script

```
$text = 'lorem ipsum elit congue auctor inceptos taciti suscipit
tortor auctor integer congue hac nullam hac auctor tellus nullam
inceptos nullam integer tellus nullam auctor elit lorem ipsum elit';
```

Add code to your script that uses an associative array `$dict3` to count the number of occurrences of each word in this string and then prints a list of all the different words and their number of occurrences.

If your code is correct, the output will be something like

```
lorem -> 2
ipsum -> 2
elit -> 3
congue -> 2
auctor -> 4
inceptos -> 2
taciti -> 1
suscipit -> 1
tortor -> 1
integer -> 2
hac -> 2
nullam -> 4
tellus -> 2
```

d. Modify the code that prints out the key-value pairs in `$dict3` so that the key-value pairs are shown in order of the values starting with the highest value. The output should take the form of a HTML table similar to the following:

Word	No of Occurrences
nullam	4
auctor	4
elit	3
tellus	2
ipsum	2
lorem	2
integer	2
hac	2
congue	2
inceptos	1
taciti	1
inceptos	1
suscipit	1
tortor	1

Hint: See the PHP Manual on array sorting at
http://php.net/manual/en/array.sorting.php.

3. In the lectures we covered the way 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. Use a text editor to create a file `php03C.php` in your `public_html` directory with the following content:

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
    <title>PHP 03C</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";
  // $a is a PHP global variable
  $a = 0;

  if ($a == 0) {
     // $b is also a PHP 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>
```

Make sure that the file permissions of that file are correct.

b. Open a web browser and access the URL

`https://student.csc.liv.ac.uk/~<user>/php03C.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 about where declarations like this need to be placed?

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

a. Use a text editor to create a file `php03D.php` in your `public_html` directory with the following content:

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
    <title>PHP 03D</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 4a</h2>\n";
function outside1() {
   $msg = "Outside!";
   function inside1() {
      $msg = "Inside!";
      echo "(I1): $msg<br>\n";
   }
   inside1();
   echo "(O1): $msg<br>\n";
}
outside1();
?>
  </body>
</html>
```

Make sure that the file permissions of that file are correct.

b. Open a web browser and access the URL

`https://student.csc.liv.ac.uk/~`*`<user>`*`/php03D.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 3 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 code at the end of the script:

```
echo "<h2>Exercise 4d</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 see 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 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 4h<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?