COMP519 Web Programming

Lecture 22: PHP (Part 4)

Handouts

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

Functions

Functions are defined as follows in PHP:

```
function identifier($param1,&$param2, ...) {
   statements
}
```

- Functions can be placed anywhere in a PHP script but preferably they should all be placed at start of the script (or at the end of the script)
- Function names are case-insensitive
- The function name must be followed by parentheses
- A function has zero, one, or more parameters that are variables
- Parameters can be given a default value using

  `$param = const_expr`
- When using default values, any defaults must be on the right side of any parameters without defaults

Functions

Functions are defined as follows in PHP:

```
function identifier($param1,&$param2, ...) {
   statements
}
```

- The return statement

  `return value`

  can be used to terminate the execution of a function and to make *value* the return value of the function
- The return value does not have to be scalar value
- A function can contain more than one return statement
- Different return statements can return values of different types

Calling a Function

A function is called by using the function name followed by a list of arguments in parentheses

```
function identifier($param1, &$param2, ...) {
   ...
}
... identifier(arg1, arg2,...) ...
```

- The list of arguments can be shorter as well as longer as the list of parameters
- If it is shorter, then default values must have been specified for the parameters without corresponding arguments
- If no default values are available for 'missing' arguments, then we get a PHP fatal error

Example:

```
function sum($num1,$num2) { return $num1+$num2; }
echo "sum: ",sum(5,4),"\n";
$sum = sum(3,2);
```

Variables and Scope

PHP distinguishes the following categories of variables with respect to scope:

- Local variables are only valid in the part of the code in which they are introduced
- (PHP) Global variables are valid outside of functions
- Superglobal variables are built-in variables that are always available in all scopes
- Static variables are local variables within a function that retain their value between separate calls of the function

By default,

- variables inside a function definition are local but not static
- variables outside any function definition are global

Functions and Scope

```
$x = "Hello";

function f1() {
  echo "1: $x\n";
}

function f2() {
  echo "2: $x\n";
  $x = "Bye";
  echo "3:",$x,"\n";
}

f1();
f2();
echo "4: $x\n";
```

- A variable defined outside any function is (PHP) global
- A (PHP) global variable can be accessed from any part of the script that is not inside a function
- A variable within a PHP function is by default local and can only be accessed within that function
- There is no hoisting of variable 'declarations'

```
1: PHP Notice:  Undefined variable: x
2: PHP Notice:  Undefined variable: x
3: Bye
4: Hello
```

Functions and Global Variables

```
$x = "Hello";

function f1() {
  global $x;
  echo "1: $x\n";
}
function f2() {
  $x = "Bye";
  echo "2: $x\n";
  global $x;
  echo "3: $x\n";
}
f1();
f2();
echo "4: $x\n";
```

```
1: Hello
2: Bye
3: Hello
4: Hello
```

- A 'local' variable can be declared to be (PHP) global using the keyword `global`
- All (PHP) global variables with the same name refer to the same storage location/data structure
- An `unset` operation removes a specific variable, but leaves other (global) variables with the same name unchanged

Functions and Global Variables

```
$x = "Hello";

function f2() {
  $x = "Bye";
  echo "1: $x\n";
  global $x;
  $x = "Hola";
  echo "2: $x\n";
  unset($x);
  echo "3: $x\n";
  $x = "Adios"
  echo "4: $x\n";
}
f2();
echo "5: $x\n";
```

- A 'local' variable can be declared to be (PHP) global using the keyword `global`
- All (PHP) global variables with the same name refer to the same storage location/data structure
- An `unset` operation removes a specific variable, but leaves other (global) variables with the same name unchanged

```
1: Bye
2: Hola
3: PHP Notice: Undefined variable: x
4: Adios
5: Hola
```

Functions and Scope (2)

```
$x = "Hello";


function f3($x) {
  $x .= '!';
  echo "1: $x\n";
}
f3('Bye');
echo "2: $x\n";
f3($x)
echo "3: $x\n";
```

```
1: Bye!
2: Hello
1: Hello!
3: Hello
```

- Parameters are local variables unrelated to any PHP global variables of the same name

PHP Functions: Example

```
function bubble_sort($array) {
  // $array, $size, $i, $j are all local
  if (!is_array($array))
     throw new Exception("Argument not an array");
  $size = count($array);
  for ($i=0; $i<$size; $i++) {
    for ($j=0; $j<$size-1-$i; $j++) {
      if ($array[$j+1] < $array[$j]) {
        swap($array, $j, $j+1);  }  }  }
  return $array;
}

function swap(&$array, $i, $j) {
  // swap gets a reference (to an array)
  $tmp = $array[$i];
  $array[$i] = $array[$j];
  $array[$j] = $tmp;
}
```

Note: The functions are not nested

PHP Functions: Example

```
function bubble_sort($array) {
  ... swap($array, $j, $j+1); ...
  return $array;
}

function swap(&$array, $i, $j) {
  $tmp = $array[$i];
  $array[$i] = $array[$j];
  $array[$j] = $tmp; }

$array = array(2,4,3,9,6,8,5,1);
echo "Before␣sorting␣", join(",␣",$array), "\n";
Before sorting 2, 4, 3, 9, 6, 8, 5, 1
```

```
$sorted = bubble_sort($array);
echo "After␣␣sorting␣", join(",␣",$array), "\n";
echo "Sorted␣array␣␣␣", join(",␣",$sorted), "\n";
After  sorting 2, 4, 3, 9, 6, 8, 5, 1
Sorted array   1, 2, 3, 4, 5, 6, 8, 9
```

Nested Functions

- PHP allows the definition of nested functions

```
function outer($param1, &$param2, ...) {
  function inner($param3, &$param4, ...) { ... }
}
```

- The inner function does not have access to local variables of the outer function
- The inner function can be called from outside the outer function
- The inner function is created the first time the outer function is called
- Calling the outer function twice will attempt to create the inner function twice

  ⇝ leads to an error that can be avoided by using

```
if (!function_exists('inner')) {
  function inner($param3, &$param4, ...) { ... }
}
```

PHP Functions and Static Variables

- A variable is declared to be static using the keyword `static` and should be combined with the assignment of an initial value (initialisation)

```
function counter() { static $count = 0; return $count++; }
```

  ⇝ static variables are initialised only once

```
function counter() { static $count = 0; return $count++; }
$count = 5;
echo "1: global \$count = $count\n";
1: global $count = 5
echo "2: static \$count = ",counter(),"\n";
echo "3: static \$count = ",counter(),"\n";
2: static $count = 0
3: static $count = 1
echo "4: global \$count = $count\n";
4: global $count = 5
echo "5: static \$count = ",counter(),"\n";
5: static $count = 2
```

Functions and HTML

- It is possible to include HTML markup in the body of a function definition
- The HTML markup can in turn contain PHP scripts
- A call of the function will execute the PHP scripts, insert the output into the HTML markup, then output the resulting HTML markup

```
<?php
function print_form($fn, $ln) {
?>
<form action="process_form.php" method=POST">
<label>First Name: <input type="text" name="f" value="<?php echo $fn ?>">
</label><br>
<label>Last Name<b>*</b>:<input type="text" name="l" value="<?php echo $ln ?>">
</label><br>
<input type="submit" name="submit" value="Submit"> <input type=reset></form>
<?php
}
print_form("Peter","Pan");
?>
```

```
<form action="process_form.php" method=POST">
<label>First Name: <input type="text" name="f" value="Peter"></label><br>
<label>Last Name<b>*</b>:<input type="text" name="l" value="Pan"></label><br>
<input type="submit" name="submit" value="Submit"> <input type=reset></form>
```

Functions with Variable Number of Arguments

The number of arguments in a function call is allowed to exceed the number of parameters of the function

$\rightsquigarrow$ the parameter list only specifies the minimum number of arguments

- `int func_num_args()`
  returns the number of arguments passed to a function
- `mixed func_get_arg(arg_num)`
  returns the specified argument, or FALSE on error
- `array func_get_args()`
  returns an array with copies of the arguments passed to a function

```
function sum() { // no minimum number of arguments
  if (func_num_args() < 1) return 0;
  $sum = 0;
  foreach (func_get_args() as $value) { $sum += $value; }
  return $sum;
}
```

Functions with Variable Number of Arguments

- Since PHP 5.6, we can use the `...` token in an argument list to denote that the function accepts a variable number of arguments
- The arguments will be passed into the given variable as an array

```php
function sum(...$numbers) {
  if (count($numbers) < 1) return 0;
  $sum = 0;
  foreach ($numbers as $value) { $sum += $value; }
  return $sum;
}
echo "1: ",sum(0,1,2,3), "\n";

echo "2: ",sum(0,TRUE,"2",3e0), "\n";
```

```
1: 6
2: 6
```

Closures and Anonymous Functions

- PHP supports anonymous functions as objects of the closure class
- Anonynous functions can be treated like any other value, e.g., stored in variables, passed as function arguments

```
$div = function($x,$y) { return $x / $y; };
```

- Via a `use` clause, anonymous functions can gain access to external variables

```
$y = 3
$funcs = [function($x) use($y) { return 2*($x+$y); },
          function($x) use($y) { return 3+($x+$y); }];
foreach ($funcs as $func) {
  echo $func(5),"\n";
}
```

```
16
11
```

Functions as Arguments

PHP allows function names and anonymous functions to be passed as arguments to other functions

```
function apply($f,$x,$y) {
  return $f($x,$y);
}
function mult($x,$y) {
  return $x * $y;
}
$div = function($x,$y) { return $x / $y; };

echo "2 * 3 = ",apply('mult',2,3),"\n";
```

```
2 * 3 = 6
```

```
echo "6 / 2 = ",apply($div,  6,2),"\n";
```

```
6 / 2 = 3
```

```
echo "2 + 3 = ",apply(function($x,$y) { return $x + $y; },
                      2,3),"\n";
```

```
2 + 3 = 5
```

Type Declarations

- PHP 5 introduced type declarations for function parameters
- PHP 7 added type declarations for the return value of a function
- By default, type juggling is still applied
- To enforce strict type checking the declaration

```
declare(strict_types=1);
```

must be added at the start of the PHP file

```
function mult5(int $x):int { return 5*$x;}
echo "1: ", mult5(1),      "\n";
echo "2: ", mult5("1.6"),   "\n";
echo "3: ", mult5([1,2]), "\n";
```

```
// without strict_types=1
1: 5
2: 5  // not 8 nor 10

PHP Fatal error:
    Uncaught TypeError
```

```
// with strict_types=1
1: 5
PHP Fatal error:
    Uncaught TypeError
PHP Fatal error:
    Uncaught TypeError
```

Including and Requiring Files

- It is often convenient to build up libraries of function definitions, stored in one or more files, that are then reused in PHP scripts
- PHP provides the commands `include`, `include_once`, `require`, and `require_once` to incorporate the content of a file into a PHP script

```
include 'mylibrary.php';
```

- PHP code in a library file must be enclosed within a PHP start tag <?php and an end PHP tag ?>
- The incorporated content inherits the scope of the line in which an `include` command occurs
- If no absolute or relative path is specified, PHP will search for the file
  - first, in the directories in the include path `include_path`
  - second, in the script's directory
  - third, in the current working directory

Including and Requiring Files

- Several `include` or `require` commands for the same library file results in the file being incorporated several times
  $\rightsquigarrow$ defining a function more than once results in an error
- Several `include_once` or `require_once` commands for the same library file results in the file being incorporated only once
- If a library file requested by `include` and `include_once` cannot be found, PHP generates a warning but continues the execution of the requesting script
- If a library file requested by `require` and `require_once` cannot be found, PHP generates a error and stops execution of the requesting script

PHP Libraries: Example

mylibrary.php

```
<?php
function bubble_sort($array) {
  ... swap($array, $j, $j+1); ...
  return $array;
}

function swap(&$array, $i, $j) {
  ...
}
?>
```

example.php

```
<?php
require_once 'mylibrary.php';
$array = array(2,4,3,9,6,8,5,1);
$sorted = bubble_sort($array);
?>
```

Revision and Further Reading

- Read
  - Chapter 5: PHP Functions and Objects: PHP Functions

  of R. Nixon: Learning PHP, MySQL & JavaScript: with jQuery, CSS & HTML5. O'Reilly, 2018.
- Read
  - Language Reference: Functions
    `http://uk.php.net/manual/en/language.functions.php`

  of P. Cowburn (ed.): PHP Manual. The PHP Group, 25 Oct 2019. `http://uk.php.net/manual/en` [accessed 26 Oct 2019]