COMP284 Scripting Languages
Lecture 11: PHP (Part 3)
Handouts (8 on 1)

Ullrich Hustadt

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

Contents

1. Special types
   NULL
   Resources
2. Control structures
   Conditional statements
   Switch statements
   While- and Do While-loops
   For-loops
3. Functions
   Defining a function
   Calling a function
   Variables

NULL

- NULL is both a special type and a valu
- NULL is the only value of type NULL
  and the name of this constant is case-insensitive
- A variable has both type NULL and value NULL in the following three situations:
  1. The variable has not yet been assigned a value (not equal to NULL)
  2. The variable has been assigned the value NULL
  3. The variable has been unset using the unset operation
- There are a variety of functions that can be used to test whether a variable is NULL including:
  - `bool isset($variable)`
    TRUE iff *$variable* exists and does not have value NULL
  - `bool is_null(expr)`
    TRUE iff *expr* is identical to NULL

NULL

Warning: Using NULL with == may lead to counter-intuitive results

```
$d = array();
echo var_dump($d), "\n";
array(0) {
}
echo 'is_null($d):␣', (is_null($d)) ? "TRUE\n": "FALSE\n";
is_null($d): FALSE
echo '$d␣===␣null:␣', ($d === null) ? "TRUE\n": "FALSE\n";
$d === null: FALSE
echo '$d␣␣==␣null:␣', ($d == null) ? "TRUE\n": "FALSE\n";
$d  == null: TRUE
```

Type juggling means that an empty array is (loosely) equal to NULL
but not identical (strictly equal) to NULL

Resources

A resource is a reference to an external resource and corresponds to a Perl filehandle

- `resource fopen(filename, mode)`
  Returns a file pointer resource for *filename* access using *mode* on success, or FALSE on error

Mode	Operation	Create	Truncate
'r'	read file		
'r+'	read/write file		
'w'	write file	yes	yes
'w+'	read/write file	yes	yes
'a'	append file	yes	
'a+'	read/append file	yes	
'x'	write file	yes	
'x+'	read/write file	yes	

See `http://www.php.net/manual/en/resource.php` for further details

Resources

- `bool fclose(resource)`
  - Closes the resource
  - Returns TRUE on success
- `string fgets(resource [, length])`
  - Returns a line read from *resource* and
    returns FALSE if there is no more data to be read
  - With optional argument *length*, reading ends when *length* − 1 bytes have been read, or a newline or on EOF (whichever comes first)
- `string fread(resource,length)`
  - Returns *length* characters read from *resource*

```
$handle = fopen('somefile.txt', 'r');
while ($line = fgets($handle)) {
```

Resources

- `[, length])`
  - *length* bytes have been written or the end of string is reached, whichever comes first
- `int fprintf(resource, format, arg1, arg2, ...)`
  - Writes a list of arguments to a resource in the given format
  - Identical to `fprintf` with output to *resource*
- `int vfprintf (resource, format, array)`
  - Writes the elements of an array to a resource in the given format
  - Identical to `vprintf` with output to *resource*

```
$handle = fopen('somefile.txt', 'w');
fwrite($handle,"Hello World!".PHP_EOL); // `logical newline'
fclose($handle);
```

In contrast to Perl, in PHP `\n` always represents the character with ASCII code 10 not the platform dependent newline ⇝ use `PHP_EOL` instead

Control structures: conditional statements

The general format of conditional statements is very similar but not identical to that in Java and Perl:

```
if (condition) {
    statements
} elseif (condition) {
    statements
} else {
    statements
}
```

- the elseif-clauses is optional and there can be more than one
  Note: `elseif` instead of `elsif`!
- the else-clause is optional but there can be at most one
- in contrast to Perl, the curly brackets can be omitted if there is only a single statement in a clause

Control structures: conditional statements/expressions

- PHP allows to replace curly brackets with a colon : combined with an endif at the end of the statement:

```
if (condition):
    statements
elseif (condition):
    statements
else:
    statements
endif
```

  This also works for the switch statement in PHP

  However, this syntax becomes difficult to parse
  when nested conditional statements are used and is best avoided

- PHP also supports conditional expressions

```
condition ? if_true_expr : if_false_expr
```

Control structures: switch statement

A switch statement in PHP takes the following form

```
switch (expr) {
  case expr1:
      statements
      break;
  case expr2:
      statements
      break;
  default:
      statements
      break;
}
```

- there can be arbitrarily many case-clauses
- the default-clause is optional but there can be at most one
- *expr* is evaluated only once and then compared to *expr1*, *expr2*, etc using (loose) equality ==
- once two expressions are found to be equal the corresponing clause is executed
- if none of *expr1*, *expr2*, etc are equal to *expr*, then the default-clause will be executed
- break 'breaks out' of the switch statement
- if a clause does not contain a break command, then executio

Control structures: switch statement

Example:

```
switch ($command) {
  case "North":
     $y += 1; break;
  case "South":
     $y -= 1; break;
  case "West";
     $x -= 1; break;
  case "East":
     $x += 1; break;
  case "Search":
     if (($x = 5) && ($y = 3))
        echo "Found a treasure\n";
      else
        echo "Nothing here\n";
      break;
  default:
     echo "Not a valid command\n"; break;
}
```

Control structures: switch statement

Not every case-clause needs to have associated statements

Example:

```
switch ($month) {
  case 1:   case 3:   case 5:   case 7:
  case 8:   case 10:  case 12:
     $days = 31;
     break;
  case 4:   case 6:   case 9:   case 11:
     $days = 30;
     break;
  case 2:
     $days = 28;
     break;
  default:
     $days = 0;
     break;
}
```

Control structures: while- and do while-loops

- PHP offers while-loops and do while-loops

```
while (condition) {
   statements
}

do {
   statements
} while (condition);
```

- As usual, curly brackets can be omitted if the loop consists of only one statement

Example:

```
// Compute the factorial of $number
$factorial = 1;
do {
     $factorial *= $number--;
} while ($number > 0);
```

Control structures: for-loops

- for-loops in PHP take the form

```
for (initialisation; test; increment) {
    statements
}
```

  Again, the curly brackets are not required if the body of the loop only consists of a single statement

- In PHP *initialisation* and *increment* can consist of more than one statement, separated by commas instead of semicolons

Example:

```
for ($i = 3, $j = 3; $j >= 0; $i++, $j--)
    echo "$i␣-␣$j␣-␣", $i*$j, "\n";
```

```
3 - 3 - 9
4 - 2 - 8
```

Control structures: break and continue

- T [illegible] o while-, and for-loops
  a [illegible]

```
w [illegible]
  $written = fwrite($resource,$value);
  if (!$written) break;
}
```

- The continue command stops the execution of the current iteration of a loop and moves the execution to the next iteration

```
for ($x = -2; $x <= 2; $x++) {
   if ($x == 0) continue;
   printf("10␣/␣%2d␣=␣%3d\n",$x,(10/$x));
}
```

```
10 / -2 =  -5
10 / -1 = -10
10 /  1 =  10
10 /  2 =   5
```

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

Example:

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

Variables

PHP distinguishes three categories of variables:

- Local variables are only accessible in the part of the code in which they are introduced
- Global variables are accessible everywhere in the code
- Static variables are local variables within a function that retain their value between separate calls of the function

By default, variables in PHP are local but not static
(Variables in Perl are by default global)

PHP functions: Example

```
function bubble_sort($array) {
  // $array, $size, $i, $j are all local
  if (!is_array($array))
     trigger_error("Argument␣not␣an␣array\n", E_USER_ERROR);
  $size = count($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 expects a reference (to an array)
  $tmp = $array[$i];
  $array[$i] = $array[$j];
  $array[$j] = $tmp;
}
```

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";
$sorted = bubble_sort($array);
echo "After  sorting ", join(", ",$array), "\n";
echo "Sorted array   ", join(", ",$sorted), "\n";
```

```
Before sorting 2, 4, 3, 9, 6, 8, 5, 1
After  sorting 2, 4, 3, 9, 6, 8, 5, 1
Sorted array   1, 2, 3, 4, 5, 6, 8, 9
```

Functions and global variables

- A variable is declared to be global using the keyword `global`

```
function echo_x($x) {
  echo $x," ";
  global $x;
  echo $x;
}

$x = 5;      // this is a global variable called $x
echo_x(10); // prints first '10' then '5'
```

⇝ an otherwise local variable is made accessible outside its normal scope using `global`

⇝ all global variables with the same name refer to the same storage location/data structure

an `unset` operation removes a specific variable, but leaves other ame unchanged

PHP functions and Global variables

```
func
  g
  i
  i
}
$x = 2; $y = 3; $z = 4;
echo "1: \$x = $x, \$y = $y, \$z = $z\n";
```

```
1: $x = 2, $y = 3, $z = 4
```

```
unset($z);
echo "2: \$x = $x, \$y = $y, \$z = $z\n";
```

```
PHP Notice:  Undefined variable: z in script on line 9
2: $x = 2, $y = 3, $z =
```

```
modify_or_destroy_var(false);
echo "3: \$x = $x, \$y = $y\n";
```

```
3: $x = 6, $y = 3
```

```
modify_or_destroy_var(true);
echo "4: \$x = $x, \$y = $y\n";
```

```
PHP Notice:  Undefined variable: x in script on line 4
4: $x = 6, $y = 3
```

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

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

```
1: global $count = 5
2: static $count = 0
3: static $count = 1
4: global $count = 5
```

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("Ullrich","Hustadt");
?>
```

```
<form action="process_form.php" method=POST">
<label>First Name: <input type="text" name="f" value="Ullrich"></label><br>
<label>Last Name<b>*</b>:<input type="text" name="l" value="Hustadt"></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 its parameters
⇝ 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 null;
  $sum = 0;
  foreach (func_get_args() as $value) { $sum += $value; }
  return $sum;
}
```

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
  ⇝ 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

Read

- Chapter 4: Expressions and Control Flow in PHP
- Chapter 5: PHP Functions and Objects
- Chapter 7: Practical PHP

of

R. Nixon:
Learning PHP, MySQL, and JavaScript.
O'Reilly, 2009.

- `http://uk.php.net/manual/en/language.control-structures.php`
- `http://uk.php.net/manual/en/language.functions.php`
- `http://uk.php.net/manual/en/function.include.php`
- `http://uk.php.net/manual/en/function.include-once.php`
  `function.require.php`
  `function.require-once.php`