

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

Special types

Resources

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

COMP284 Scripting Languages

Lecture 11

Slide L11 – 4

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

COMP284 Scripting Languages

Lecture 11

Slide L11 – 3

Special types

Resources

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($handle)) {  
    // processing the line of the file  
}  
fclose($handle);
```

COMP284 Scripting Languages

Lecture 11

Slide L11 – 5

NULL

- **NULL** is both a **special type** and a **value**
- **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**

COMP284 Scripting Languages

Lecture 11

Slide L11 – 2

Resources

- **int** **fwrite**(*resource*, *string* [, *length*])

- Writes a *string* to a resource
- If *length* is given, writing stops after *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** **fprintf** (*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 `\n` use `PHP_EOL` instead

COMP284 Scripting Languages

Lecture 11

Slide L11 – 6

NULL

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

```
$d = array();  
echo var_dump($d), "\n";  
array() {  
}  
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**

COMP284 Scripting Languages

Lecture 11

Slide L11 – 3

Control structures

Conditional statements

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

COMP284 Scripting Languages

Lecture 11

Slide L11 – 7

<div>Control structures</div> <div>Conditional statements</div> <div>Control structures: conditional statements/expressions</div> <ul style="list-style-type: none"> PHP allows to replace curly brackets with a colon : combined with an endif at the end of the statement: <pre>if (condition): statements elseif (condition): statements else: statements endif</pre> <p>This also works for the switch statement in PHP</p> <p>However, this syntax becomes difficult to parse when nested conditional statements are used and is best avoided</p> <ul style="list-style-type: none"> PHP also supports conditional expressions <pre>condition ? if_true_expr : if_false_expr</pre> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 8</div>	<div>Control structures</div> <div>While- and Do While-loops</div> <div>Control structures: while- and do while-loops</div> <ul style="list-style-type: none"> PHP offers while-loops and do while-loops <pre>while (condition) { statements } do { statements } while (condition);</pre> <p>As usual, curly brackets can be omitted if the loop consists of only one statement</p> <p>Example:</p> <pre>// Compute the factorial of \$number \$factorial = 1; do { \$factorial *= \$number--; } while (\$number > 0);</pre> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 12</div>
<div>Control structures</div> <div>Switch statements</div> <div>Control structures: switch statement</div> <p>A switch statement in PHP takes the following form</p> <pre>switch (expr) { case expr1: statements break; case expr2: statements break; default: statements break; }</pre> <ul style="list-style-type: none"> 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 corresponding 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 execution moves to the next clause <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 9</div>	<div>Control structures</div> <div>For-loops</div> <div>Control structures: for-loops</div> <ul style="list-style-type: none"> for-loops in PHP take the form <pre>for (initialisation; test; increment) { statements }</pre> <p>Again, the curly brackets are not required if the body of the loop only consists of a single statement</p> <ul style="list-style-type: none"> In PHP initialisation and increment can consist of more than one statement, separated by commas instead of semicolons <p>Example:</p> <pre>for (\$i = 1; \$i <= 5; \$i++) { echo "Sum: \$i + \$i = " . (\$i + \$i) . "\n"; }</pre> <pre>3 - 3 = 9 4 - 2 = 8 5 - 1 = 5 6 - 0 = 0</pre> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 13</div>
<div>Control structures</div> <div>Switch statements</div> <div>Control structures: switch statement</div> <p>Example:</p> <pre>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; }</pre> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 10</div>	<div>Control structures</div> <div>For-loops</div> <div>Control structures: break and continue</div> <ul style="list-style-type: none"> The break command can also be used in while-, do while-, and for-loops and it continues the execution of the loop <pre>while (\$value = array_shift(\$data) { \$written = fwrite(\$resource, \$value); if (!\$written) break; }</pre> The continue command stops the execution of the current iteration of a loop and moves the execution to the next iteration <pre>for (\$x = -2; \$x <= 2; \$x++) { if (\$x == 0) continue; printf("10 / %2d = %3d\n", \$x, (10/\$x)); }</pre> <pre>10 / -2 = -5 10 / -1 = -10 10 / 1 = 10 10 / 2 = 5</pre> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 14</div>
<div>Control structures</div> <div>Switch statements</div> <div>Control structures: switch statement</div> <p>Not every case-clause needs to have associated statements</p> <p>Example:</p> <pre>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; }</pre> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 11</div>	<div>Functions</div> <div>Defining a function</div> <div>Functions</div> <p>Functions are defined as follows in PHP:</p> <pre>function identifier(\$param1, &\$param2, ...) { statements }</pre> <ul style="list-style-type: none"> 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 <pre>\$param = const_expr</pre> When using default values, any defaults must be on the right side of any parameters without defaults <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 15</div>

<div> <div>Functions</div> <div>Defining a function</div> <div> <h2>Functions</h2> <p>Functions are defined as follows in PHP:</p> <pre>function identifier(\$param1,&\$param2, ...) { statements }</pre> <ul style="list-style-type: none"> 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 </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 16</div> </div> </div>	<div> <div>Functions</div> <div>Variables</div> <div> <h2>PHP functions: Example</h2> <pre>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</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 20</div> </div> </div>
<div> <div>Functions</div> <div>Calling a function</div> <div> <h2>Calling a function</h2> <p>A function is called by using the function name followed by a list of arguments in parentheses</p> <pre>function identifier(\$param1, &\$param2, ...) { ... } ... identifier(arg1, arg2,...) ...</pre> <ul style="list-style-type: none"> 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 <p>Example:</p> <pre>function sum(\$num1, \$num2) { return \$num1+\$num2; } echo "sum: ", sum(5,4), "\n"; \$sum = sum(3,2);</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 17</div> </div> </div>	<div> <div>Functions</div> <div>Variables</div> <div> <h2>Functions and global variables</h2> <ul style="list-style-type: none"> A variable is declared to be global using the keyword global <pre>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'</pre> <ul style="list-style-type: none"> 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 (global) variables with the same name unchanged </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 21</div> </div> </div>
<div> <div>Functions</div> <div>Variables</div> <div> <h2>Variables</h2> <p>PHP distinguishes three categories of variables:</p> <ul style="list-style-type: none"> 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 <p>By default, variables in PHP are local but not static (Variables in Perl are by default global)</p> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 18</div> </div> </div>	<div> <div>Functions</div> <div>Variables</div> <div> <h2>PHP functions and Global variables</h2> <pre>function modify_or_destroy_var(\$arg) { global \$x, \$y; if (is_bool(\$arg) && !\$arg) { \$x = \$x * \$y; } if (is_bool(\$arg) && \$arg) { unset(\$x); echo \$x; } } \$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</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 22</div> </div> </div>
<div> <div>Functions</div> <div>Variables</div> <div> <h2>PHP functions: Example</h2> <pre>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; }</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 19</div> </div> </div>	<div> <div>Functions</div> <div>Variables</div> <div> <h2>PHP functions and Static variables</h2> <ul style="list-style-type: none"> A variable is declared to be static using the keyword static and should be combined with the assignment of an initial value (initialisation) <pre>function counter() { static \$count = 0; return \$count++; }</pre> <ul style="list-style-type: none"> static variables are initialised only once <pre>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</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 23</div> </div> </div>

<div> <div>Functions</div> <div>Functions and HTML</div> <div> <h2>Functions and HTML</h2> <ul style="list-style-type: none"> 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 <pre><?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>
 <label>Last Name<input type="text" name="l" value="<?php echo \$ln?>"></label>
 <input type="submit" name="submit" value="Submit"> <input type="reset"> </form> <?php } print_form("Ullrich", "Hustadt"); ?></pre> <pre><form action="process_form.php" method="POST"> <label>First Name: <input type="text" name="f" value="Ullrich"></label>
 <label>Last Name<input type="text" name="l" value="Hustadt"></label>
 <input type="submit" name="submit" value="Submit"> <input type="reset"> </form></pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 24</div> </div> </div>	<div> <div>PHP libraries</div> <div>Include/Require</div> <div> <h2>PHP Libraries: Example</h2> <pre>mylibrary.php <?php function bubble_sort(\$array) { ... swap(\$array, \$j, \$j+1); ... return \$array; } function swap(&\$array, \$i, \$j) { ... } ?></pre> <pre>example.php <?php require_once 'mylibrary.php'; \$array = array(2,4,3,9,6,8,5,1); \$sorted = bubble_sort(\$array); ?></pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 28</div> </div> </div>
<div> <div>Functions</div> <div>Variable-length argument lists</div> <div> <h2>Functions with variable number of arguments</h2> <p>The number of arguments in a function call is allowed to exceed the number of its parameters</p> <p>~ the parameter list only specifies the minimum number of arguments</p> <ul style="list-style-type: none"> int func_num_args() returns the number of arguments passed to a function mixed func_get_arg(<i>arg_num</i>) returns the specified argument, or FALSE on error array func_get_args() returns an array with copies of the arguments passed to a function <pre>function sum() { // func_num_args() returns number of arguments if (func_num_args() < 1) return null; \$sum = 0; foreach (func_get_args() as \$value) { \$sum += \$value; } return \$sum; }</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 25</div> </div> </div>	<div> <div>PHP libraries</div> <div>Include/Require</div> <div> <h2>Revision</h2> <p>Read</p> <ul style="list-style-type: none"> Chapter 4: Expressions and Control Flow in PHP Chapter 5: PHP Functions and Objects Chapter 7: Practical PHP <p>of</p> <p>R. Nixon: Learning PHP, MySQL, and JavaScript. O'Reilly, 2009.</p> <ul style="list-style-type: none"> 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 http://uk.php.net/manual/en/function.require.php http://uk.php.net/manual/en/function.require-once.php </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 29</div> </div> </div>
<div> <div>PHP libraries</div> <div>Include/Require</div> <div> <h2>Including and requiring files</h2> <ul style="list-style-type: none"> 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 <pre>include 'mylibrary.php';</pre> PHP code in a library file must be enclosed within a PHP start tag <code><?php</code> and an end PHP tag <code>?></code> 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 <ul style="list-style-type: none"> first, in the directories in the include path <code>include_path</code> second, in the script's directory third, in the current working directory </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 26</div> </div> </div>	
<div> <div>PHP libraries</div> <div>Include/Require</div> <div> <h2>Including and requiring files</h2> <ul style="list-style-type: none"> Several include or require commands for the same library file results in the file being incorporated several times <p>~ defining a function more than once results in an error</p> 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 </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 11</div> <div>Slide L11 – 27</div> </div> </div>	