COMP284 Scripting Languages
Lecture 10: PHP (Part 2)
Handouts

Ullrich Hustadt

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

Contents

Integers and Floating-point numbers

- PHP distinguishes between
  - integer numbers 0 2012 -40 1263978
  - floating-point numbers 1.23 256.0 -12e3 2.1e-10
- PHP supports a wide range of pre-defined mathematical functions

`abs(`*number*`)`	absolute value
`ceil(`*number*`)`	round fractions up
`floor(`*number*`)`	round fractions down
`round(`*number* `[,`*prec*`,`*mode*`])`	round fractions
`log(`*number* `[,`*base*`])`	logarithm
`rand(`*min*`,`*max*`)`	generate an integer random number
`sqrt(`*number*`)`	square root

- PHP provides a range of pre-defined number constants including

`M_PI`	3.14159265358979323846
`NAN`	'not a number'
`INF`	'infinity'

Integers and Floating-point numbers: NAN and INF

The constants `NAN` and `INF` are used as return values for some applications of mathematical functions that do not return a number

- `log(0)` returns `-INF` (negative 'infinity')
- `sqrt(-1)` returns `NAN` ('not a number')

In contrast

- `1/0` returns `FALSE` and produces an error message
- `0/0` returns `FALSE` and produces an error message

and execution of the script continues!

In PHP 7

- `1/0` returns `INF` and produces an error message
- `0/0` returns `NAN` and produces an error message

and execution of the script continues!

Integers and Floating-point numbers: NAN and INF

NAN and INF can be compared with each other and other numbers using equality and comparison operators:


```
NAN == NAN ⇝ FALSE   | NAN === NAN ⇝ FALSE  | NAN == 1 ⇝ FALSE
INF == INF ⇝ FALSE   | INF === INF ⇝ TRUE   | INF == 1 ⇝ FALSE
NAN < NAN  ⇝ TRUE    | INF < INF   ⇝ TRUE   | 1 < INF  ⇝ TRUE
NAN < INF  ⇝ TRUE    | INF < NAN   ⇝ TRUE   | INF < 1  ⇝ FALSE
NAN < 1    ⇝ TRUE    | 1 < NAN     ⇝ TRUE
```


In PHP 5.3 and earlier versions, INF == INF returns FALSE

In PHP 5.4 and later versions, INF == INF returns TRUE

Integers and Floating-point numbers: NAN and INF

- PHP provides three functions to test whether a value is or is not `NAN`, `INF` or `-INF`:
  - `bool is_nan(value)`
    returns TRUE iff *value* is `NAN`
  - `bool is_infinite(value)`
    returns TRUE iff *value* is `INF` or `-INF`
  - `bool is_finite(value)`
    returns TRUE iff *value* is neither `NAN` nor `INF`/`-INF`
- In conversion to a boolean value,
  both `NAN` and `INF` are converted to `TRUE`
- In conversion to a string,
  `NAN` converts to `'NAN'` and `INF` converts to `'INF'`

Exceptions and error handling

PHP distinguishes between exceptions and errors

- A possible way to perform exception handling in PHP is as follows:

```
try {  ... run code here ...              // try
} catch (Exception $e) {
       ... handle the exception here using $e  // catch
}
```

- Errors must be dealt with by an error handling function ('Division by zero' produces an error not an exception)

  One possible approach is to let the error handling function turn errors into exceptions

```
function exception_error_handler($errno, $errstr,
  $errfile, $errline ) {
  throw new ErrorException($errstr, $errno,
                           0, $errfile, $errline); }
set_error_handler("exception_error_handler");
```

http://www.php.net/manual/en/class.errorexception.php

Booleans

- Unlike Perl, PHP does have a boolean datatype with constants `TRUE` and `FALSE` (case insensitive)
- PHP offers the same short-circuit boolean operators as Java and Perl:

  `&&` (conjunction) `||` (disjunction) `!` (negation)

  - Alternatively, `and` and `or` can be used instead of `&&` and `||`, respectively
  - However, `not` is not a PHP operator
- The truth tables for these operators are the same as for Perl
- Remember that `&&` and `||` are not commutative, that is,
  `(A && B)` is not the same as `(B && A)`
  `(A || B)` is not the same as `(B || A)`

Type conversion to boolean

When converting to boolean, the following values are considered `FALSE`:

- the boolean `FALSE` itself
- the integer 0 (zero)
- the float 0.0 (zero)
- the empty string, and the string '0'
- an array with zero elements
- an object with zero member variables (PHP 4 only)
- the special type `NULL` (including `unset` variables)
- SimpleXML objects created from empty tags

Every other value is considered `TRUE` (including any resource)

Strings

- PHP supports both single-quoted and double-quoted strings
- PHP also supports heredocs as a means to specify multi-line strings. The only difference to Perl is the use of <<< instead of << in their definition:

```
<<<identifier
here document
identifier
```

  - *identifier* might optionally be surrounded by double-quotes
  - *identifier* might also be surrounded by single-quotes, making the string a nowdoc in PHP terminology


```
print '<html>
<head><title>Multi-line String</title></head>';
print <<<EOF
<body>Some text</body>
</html>
EOF;
```

Strings

- Variable interpolation is applied to double-quoted strings (with slight differences to Perl)
- The string concatenation operator is denoted by '.' (as in Perl)
- Instead of Perl's string multiplication operator 'x' there is
  `string str_repeat(string_arg, number)`
- There are no built-in HTML shortcuts in PHP


```
$title  = "String Multiplication";
$string = "<p>I shall not repeat myself.<p>\n";
print "<!DOCTYPE html>\n<html><head><title>$title</title>
</head><body>'.str_repeat($string,3).'</body></html>';
```

```
<!DOCTYPE html>
<html><head><title>String Multiplication</title>
</head><body><p>I shall not repeat myself.<p>
<p>I shall not repeat myself.<p>
<p>I shall not repeat myself.<p>
</body></html>
```

Arrays

- PHP only supports associative arrays (hashes), simply called arrays
- PHP arrays are created using the array construct or, since PHP 5.4 [...]:

```
array(key => value, ... )
[key => value, ...]
```

where *key* is an integer or string and *value* can be of any type, including arrays

```
$arr1 = [1 => "Peter", 3 => 2009, "a" => 101];
$arr2 = array(200846369 => array("name" => "Jan␣Olsen",
                                 "COMP101" => 69,
                                 "COMP102" => 52));
```

- The size of an array can be determined using the count function:

```
int count(array [, mode])
```

```
print count($arr1);      // prints 3
print count($arr2);      // prints 1
print count($arr2,1);    // prints 4
```

Arrays

- It is possible to omit the keys when using the `array` construct:

```
$arr3 = array("Peter", "Paul", "Mary");
```

The values given in `array` will then be associated with the natural numbers 0, 1, ...

- All the keys of an array can be retrieved using

```
array_keys($array1)
```

  ⇝ returns a natural number-indexed array containing the keys of `$array1`

- All the values of an array can be retrieved using

```
array_values($array1)
```

  ⇝ returns a natural number-indexed array containing the values stored in `$array1`

Arrays

- An individual array element can be accessed via its key
- Accessing an undefined key produces an error message and returns NULL


```
$arr1 = array(1 => "Peter", 3 => 2009, "a"=> 101);

print "'a': ".$arr1["a"]."\n";
'a': 101
print "'b': ".$arr1["b"]."\n";
PHP Notice:  Undefined index: b in <file> on line <lineno>
'b':          // $arr1["b"] returns NULL
$arr1['b'] = 102;
print "'b': ".$arr1["b"]."\n";
'b': 102
```

Arrays

- PHP allows the construct

```
$array[] = value;
```


PHP will determine the maximum value M among the integer indices in $\$array$ and use the key $K = M + 1$; if there are no integer indices in $\$array$, then $K = 0$ will be used

↝ auto-increment for array keys


```
$arr4[] = 51; // 0 => 51
$arr4[] = 42; // 1 => 42
$arr4[] = 33; // 2 => 33
```


- A key-value pair can be removed from an array using the `unset` function:

```
$arr1 = array(1 => "Peter", 3 => 2009, "a" => 101);
unset($arr1[3]);    // Removes the pair 3 => 2009
unset($arr1);       // Removes the whole array
```

Arrays: foreach-loop

- PHP provides a foreach-loop construct to 'loop' through the elements of an array
- Syntax and semantics is slightly different from that of the corresponding construct in Perl


```
foreach (array as $value)
    statement

foreach (array as $key => $value)
    statement
```


- *array* is an array-expression
- $*key* and $*value* are two variables storing a different key-value pair in *array* at each iteration of the foreach-loop

- We call $*value* the foreach-variable
- foreach iterates through an array in the order in which elements were defined

Arrays: foreach-loop

`foreach` iterates through an array in the order in which elements were defined

Example 1:

```
foreach (array("Peter", "Paul", "Mary") as $key => $value)
   print "The␣array␣maps␣$key␣to␣$value\n";
```

```
The array maps 0 to Peter
The array maps 1 to Paul
The array maps 2 to Mary
```


Example 2:

```
$arr5[2] = "Marry";
$arr5[0] = "Peter";
$arr5[1] = "Paul";
// 0 => 'Peter',  1 => 'Paul',  2 => 'Marry'
foreach ($arr5 as $key => $value)
   print "The␣array␣maps␣$key␣to␣$value\n";
```



```
The array maps 2 to Mary
The array maps 0 to Peter
The array maps 1 to Paul
```

Arrays: foreach-loop

Does changing the value of the foreach-variable change the element of the list that it currently stores?

Example 3:

```
$arr6 = array("name" => "Peter", "year" => 2009);

foreach ($arr6 as $key => $value) {
    print "The array maps $key to $value\n";
    $value .= " - modified"; // Changing $value
}
print "\n";

foreach ($arr6 as $key => $value)
    print "The array now maps $key to $value\n";
```



```
The array maps name to Peter
The array maps year to 2009

The array now maps name to Peter
The array now maps year to 2009
```

Arrays: foreach-loop

- In order to modify array elements within a `foreach`-loop we need use a reference

```
foreach (array as &$value)
    statement
unset($value);

foreach (array as $key => &$value)
    statement
unset($value);
```

- In the code schemata above, `&$value` is a variable whose value is stored at the same location as an array element
- Note that PHP does not allow the key to be a reference
- The `unset` statement is important to return `$value` to being a 'normal' variable

Arrays: foreach-loop

In order to modify array elements within a foreach-loop we need use a reference

Example:


```
$arr6 = array("name" => "Peter", "year" => 2009);
foreach ($arr6 as $key => &$value) { // Note: reference!
    print "The␣array␣maps␣$key␣to␣$value\n";
    $value .= "␣-␣modified";
}
unset($value); // Remove the reference from $value
print "\n";

foreach ($arr6 as $key => $value)
    print "The␣array␣now␣maps␣$key␣to␣$value\n";
```



```
The array maps name to Peter
The array maps year to 2009

The array now maps name to Peter - modified
The array now maps year to 2009 - modified
```

Array functions

PHP has no stack or queue data structures,
but has stack and queue functions for arrays:

- `array_push($array, value1, value2, ...)`
  appends one or more elements at the end of the end of an array variable;
  returns the number of elements in the resulting array
- `array_pop($array)`
  extracts the last element from an array and returns it

- `array_shift($array)`
  shift extracts the first element of an array and returns it
- `array_unshift($array, value1, value2, ...)`
  inserts one or more elements at the start of an array variable;
  returns the number of elements in the resulting array

Note: `$array` needs to be a variable

Printing

In PHP, the default command for generating output is `echo`

- `void echo(arg1)`
- `void echo(arg1, arg2, ...)`
  - Outputs all arguments
  - No parentheses are allowed if there is more than one argument
  - More efficient than print (and therefore preferred).

Additionally, PHP also provides the functions `print`, and `printf`:

- `int print(arg)`
  - Outputs its argument
    Only one argument is allowed!
  - Returns value 1
  - Parentheses can be omitted

Printing

- `string sprintf(format, arg1, arg2, ....)`
  - Returns a string produced according to the formatting string *format*
  - Parentheses are necessary

  See `http://www.php.net/manual/en/function.sprintf.php` for details

- `int printf(format, arg1, arg2, ...)`
  - Produces output according to *format*
  - Parentheses are necessary
  - Returns the length of the outputted string

- Important: In contrast to Perl, a PHP array cannot take the place of a list of arguments

```
printf("%2d apples %2d oranges\n",array(5,7));
```

produces an error message

Printing

- `string vsprintf(format, array)`
  - Returns a string produced according to the formatting string *format*
  - Identical to `sprintf` but accepts an `array` as argument
  - Parentheses are necessary

- `int vprintf(format, array)`
  - Produces output according to *format*
  - Identical to `printf` but accepts an `array` as argument
  - Parentheses are necessary


```
vprintf("%2d apples %2d oranges\n",array(5,7));
```

```
5 apples 7 oranges
```

Revision

Read

- Chapter 6: PHP Arrays

of

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

- `http://uk.php.net/manual/en/language.types.boolean.php`
- `http://uk.php.net/manual/en/language.types.integer.php`
- `http://uk.php.net/manual/en/language.types.float.php`
- `http://uk.php.net/manual/en/language.types.string.php`
- `http://uk.php.net/manual/en/language.types.array.php`
- `http://uk.php.net/manual/en/control-structures.foreach.php`