

COMP284 Scripting Languages

Lecture 10: FHP (Part 2)

Handouts

Assignment Project Exam Help

<https://eduassistpro.github.io>

Department of Computer

School of Electrical Engineering, Electronics, and

University of Liverpool

Add WeChat edu_assist_pro

1 Scalar types

- Integers and Floating-point numbers

- Exceptions and error handling

- Bo

- Stri

2 Compound types

- Arrays

- Foreach-loops

- Array functions

3 Printing

Integers and Floating-point numbers

- PHP distinguishes between

- integer numbers 0 2012 -40 1263978

- floating-point numbers 1.25 256.9 -12e19 2.4e-10

- PHP supports a wide range of pre-defined mathematical functions

```
abs(
```

```
ceil
```

```
floo
```

```
roun
```

```
log(number [, base])
```

```
rand(min, max)
```

```
sqrt(number)
```

```
log
```

```
gen
```

```
squ
```

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

```
M_PI
```

```
NAN
```

```
INF
```

```
3.14159265358979323846
```

```
'not a number'
```

```
'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`
- `0/0` returns `FALSE` and produces an error

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!

Assignment Project Exam Help
<https://eduassistpro.github.io>
Add WeChat edu_assist_pro

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					E
NAN < INF					SE
NAN < 1					

In PHP 5.3 and earlier versions, INF == INF

In PHP 5.4 and later versions, INF == INF

Add WeChat edu_assist_pr

<https://eduassistpro.github.io>

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 if *value* is NAN

- `bool is_infinite(value)`
retu

- `bo`
retu

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

- Errors
(‘Divi

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**)

- **Alte** **actively**
- **Ho**
- The **truth tables** for these operators are the same as for Perl
- Remember that **&&** and **||** are **not** commu
 $(A \ \&\& \ B)$ is not the same as $(B \ \&\& \ A)$
 $(A \ || \ B)$ is not the same as $(B \ || \ A)$

<https://eduassistpro.github.io>

Add WeChat: edu_assist_pro

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
- an array with no elements
- an object with no properties
- the special type NULL (including **unset**)
- 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:

```
<<<id  
here docu  
identif
```

- *ide*
- *identifier* might also be surrounded by single-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

`stri`

- There is

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

```

Arrays

- PHP only supports **associative arrays** (**hashes**), simply called **arrays**
- PHP **arrays** are created using the **array** construct or, since PHP 5.4 [1]:

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

where
includ

```
$arr1 = [1 => "Pe  
$arr2 = array(200846369 => array("name" => "Jan_Olsen",
```

- The size of an array can be determined using the `count` function [2]:
`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': 101
```

```
print " "
```

```
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

```
$arr
```

```
→ au
```

```
$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 (
    s
    foreach (
        statement
```

- *array* is an array expression
- *\$key* and *\$value* are two variables storing a *key* and *value* of the *array* at each iteration of the `foreach-l`
- 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 Pe

The array maps 1 to Pa

The array maps 2 to Ma

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";  
}
```

```
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 &$amp;value)  
{  
    statement  
}  
unset($value);
```

```
foreach (  
    $  
unset(  
    $
```

- In the code schemata above, `&$value` is a `reference` to the same location as an array element.
- Note that PHP does not allow the `key` to be a reference.
- The `unset` statement is important to return `$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)  
    print "The array maps $key to $value\n";  
unset($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;
return

- `array_pop($array)`

extracts the last element of an array variable 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, ...)`

- Out
- No p
- Mor

Additionally, PHP also provides the functions

- `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 det

- `int`

<https://eduassistpro.github.io>

- 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

Add WeChat [edu_assist_pro](https://eduassistpro.github.io)

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`

- Pro

- Identical to `printf` but accepts an `array` as argu

- Parentheses are necessary

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

```
5 apples 7 oranges
```


Revision

Read

- Chapter 6: PHP Arrays of

R. Nix

Learni

O'Reilly, 2009.

- <http://nl.php.net/manual/en/language.t>
- <http://uk.php.net/manual/en/language.t>
- <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>