# COMP519 Web Programming

## Lecture 20: PHP (Part 2)

## Handouts

Ullrich Hustadt

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

# Contents

# Comparison Operators

Type juggling also plays a role in the way PHP comparison operators work:

| | | |
|---|---|---|
| *expr1* == *expr2* | Equal | TRUE iff *expr1* is equal to *expr2* after type juggling |
| *expr1* != *expr2* | Not equal | TRUE iff *expr1* is not equal to *expr2* after type juggling |
| *expr1* <> *expr2* | Not equal | TRUE iff *expr1* is not equal to *expr2* after type juggling |
| *expr1* === *expr2* | Identical | TRUE iff *expr1* is equal to *expr2*, and they are of the same type |
| *expr1* !== *expr2* | Not identical | TRUE iff *expr1* is not equal to *expr2*, or they are not of the same type |

Note: For ==, !=, and <>, numerical strings are converted to numbers and compared numerically

```
"123" == 123              ⇝  TRUE
"123" != 123              ⇝  FALSE
"1.23e2" == 123           ⇝  TRUE
"1.23e2" == "12.3e1"      ⇝  TRUE
"10hello5" == 10          ⇝  TRUE
5 == TRUE                 ⇝  TRUE
```

```
"123" === 123             ⇝  FALSE
"123" !== 123             ⇝  TRUE
1.23e2 === 123            ⇝  FALSE
"1.23e2" === "12.3e1"     ⇝  FALSE
"10hello5" === 10"        ⇝  FALSE
5 === TRUE                ⇝  FALSE
```

# Comparison Operators

Type juggling also plays a role in the way PHP comparison operators work:

| | | |
|---|---|---|
| *expr1* < *expr2* | Less than | TRUE iff *expr1* is strictly less than *expr2* after type juggling |
| *expr1* > *expr2* | Greater than | TRUE iff *expr1* is strictly greater than *expr2* after type juggling |
| *expr1* <= *expr2* | Less than or equal to | TRUE iff *expr1* is less than or equal to *expr2* after type juggling |
| *expr1* >= *expr2* | Greater than or equal to | TRUE iff *expr1* is greater than or equal to *expr2* after type juggling |

Note: For >, >=, <, and <= numerical strings are converted to numbers and compared numerically

```
'35.5' > 35              ⇝  TRUE
'ABD' > 'ABC'            ⇝  TRUE
'1.23e2' > '12.3e1'      ⇝  FALSE
"F1" < "G0"              ⇝  TRUE
TRUE > FALSE             ⇝  TRUE
5 > TRUE                 ⇝  FALSE
```

```
'35.5' >= 35             ⇝  TRUE
'ABD' >= 'ABC'           ⇝  TRUE
'1.23e2' >= '12.3e1'     ⇝  TRUE
"F1" <= "G0"             ⇝  TRUE
TRUE >= FALSE            ⇝  TRUE
5 >= TRUE                ⇝  TRUE
```

# Comparison operators

- To compare strings 'as strings' the `strcmp` function can be used
- PHP 7 introduced the so-called 'spaceship operator' for three-way comparisons (that converts numeric strings to numbers)

| | | |
|---|---|---|
| `strcmp(`*expr1*`,`*expr2*`)` | String comparison | Returns < 0 if *expr1* is less than *expr2*,<br>> 0 if *expr1* is greater than *expr2*,<br>0 if *expr1* is equal to *expr2* |
| *expr1* `<=>` *expr2*<br>(PHP 7 only) | Three-way comparison | Returns −1 if *expr1* < *expr2*,<br>+1 if *expr1* > *expr2*,<br>0 if *expr1* == *expr2* |

```
strcmp('ABD','ABC')          ⇝  1
strcmp('aaa',"aaa")          ⇝  0
strcmp('1.23e2','12.3e1')    ⇝  -1
'ABD' <=> 'ABC'              ⇝  1
'aaa' <=> "aaa"              ⇝  0
'1.23e2' <=> '12.3e1'        ⇝  0
'35.5' <=> 35                ⇝  1
TRUE <=> FALSE               ⇝  1
5 <=> TRUE                   ⇝  0
```

```
strcmp("F1","G0")            ⇝  -65536
strcmp('aaa',"AAA")          ⇝  2105376

"F1" <=> "G0"                ⇝  -1
'aaa' <=> "AAA"              ⇝  1

'10hello5' <=> 10            ⇝  0
0.0 <=> FALSE                ⇝  0
'FALSE' <=> TRUE             ⇝  0
```

# 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
INF == INF ⇝ TRUE
NAN < NAN  ⇝ TRUE
NAN < INF  ⇝ TRUE
NAN < 1    ⇝ TRUE
```

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

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

In PHP 5.3 and earlier versions, `INF == INF` returns `FALSE`
`INF === INF` returns `TRUE`
In PHP 5.4 and later versions, `INF == INF` returns `TRUE`
`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'`

# 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",
                                 "COMP518" => 69,
                                 "COMP519" => 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 a PHP notice 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
```

- The function array_key_exists(*key*, *array1*) can be used to check whether there is a value for key in *array1*

```
array_key_exists("a",$arr1)   # returns TRUE
array_key_exists("c",$arr1)   # returns FALSE
```

# 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
$\rightsquigarrow$ 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

```
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] = "Mary";
$arr5[0] = "Peter";
$arr5[1] = "Paul";
// 0 => 'Peter',  1 => 'Paul',  2 => 'Mary'
foreach ($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";
```

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

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

```
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
  - 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
- In each iteration, $value contains a reference to an array element
  ⇝ changing $value changes the array element

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

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

```
// See what the content of $arr6 is now
foreach ($arr6 as $key => $value)
    print "The␣array␣now␣maps␣$key␣to␣$value\n";
```

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

# Array Assignments

- In JavaScript arrays were objects and as a consequence array assignments were done by reference
- In PHP, this is not the case

```
$mem1 = memory_get_usage();
$array1 = range(1, 1000);
$mem2 = memory_get_usage();
echo "(1) ",sprintf("%6d",$mem2-$mem1)," more bytes\n";
$array2 = $array1;
$mem3 = memory_get_usage();
echo "(2) ",sprintf("%6d",$mem3-$mem2)," more bytes\n";
$array2[1] += 10000;
echo "\$array1[1] = ",$array1[1]," | ";
echo "\$array2[1] = ",$array2[1],"\n";
$mem4 = memory_get_usage();
echo "(3) ",sprintf("%6d",$mem4-$mem3)," more bytes\n";
```

```
(1)  36920 more bytes
(2)      0 more bytes
$array1[1] = 2 | $array2[1] = 10002
(3)  36920 more bytes
```

The PHP implementation uses copy-on-write for array assignments

# Array Assignments

- The PHP implementation uses copy-on-write for array assignments
- If we want two array variables to point to the same array literal, then we need to explicitly use a reference

```
$array1 = range(1, 1000);
$mem2 = memory_get_usage();
$array2 = &$array1;
$mem3 = memory_get_usage();
echo "(2) ",sprintf("%6d",$mem3-$mem2)," more bytes\n";
$array2[1] += 10000;
echo "\$array1[1] = ",$array1[1]," | ";
echo "\$array2[1] = ",$array2[1],"\n";
$mem4 = memory_get_usage();
echo "(3) ",sprintf("%6d",$mem4-$mem3)," more bytes\n";
```

```
(2)     24 more bytes
$array1[1] = 10002 | $array2[1] = 10002
(3)      0 more bytes
```

# Array Operators

PHP has no stack or queue data structures,
but has stack and queue operators 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

# Revision and Further Reading

- Read
  - Chapter 4: Expressions and Control Flow in PHP: Operators
  - Chapter 6: PHP Arrays

  of R. Nixon: Learning PHP, MySQL & JavaScript: with jQuery, CSS & HTML5. O'Reilly, 2018.
- Read
  - Language Reference: Types: Arrays
    `http://uk.php.net/manual/en/language.types.array.php`
  - Language Reference: Control Structures: foreach
    `http://uk.php.net/manual/en/control-structures.foreach.php`

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