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)

<table>
<thead>
<tr>
<th><code>strcmp(expr1, expr2)</code></th>
<th>String comparison</th>
<th>Returns if <code>expr1</code> is less than <code>expr2</code>, or <code>expr1</code> is greater than <code>expr2</code> or equal to <code>expr2</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expr1 &lt;= expr2</code></td>
<td>(PHP 7 only)</td>
<td><code>TRUE</code> if <code>expr1</code> is less than or equal to <code>expr2</code></td>
</tr>
<tr>
<td><code>expr1 &gt;= expr2</code></td>
<td></td>
<td><code>FALSE</code> if <code>expr1</code> is greater than or equal to <code>expr2</code></td>
</tr>
</tbody>
</table>

| `strcmp('ABD', 'ABC')` | ~ 1               | `strcmp('FI', 'GO')` | ~ 65536 |
| `strcmp('aaa', 'aab')` | ~ 0               | `strcmp('1.23e2', '12.3e1')` | ~ -1 |
| `1.23e2 < 12.3e1`     |                   | `1.23e2 < 12.3e1` | 0 |
| `true` => 5           | TRUE              | `true` => 5        | FALSE |

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 == INF` returns `NULL` if `INF == INF` returns TRUE
- `NAN < INF` returns `TRUE` if `INF < INF` returns `true`
- `NAN != INF` returns `TRUE` if `INF < INF` returns `true`
- `NAN !== INF` returns `TRUE` if `INF > INF` returns `false`

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

Arrays

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

```php
$array = ['Peter', 3, 'a' => 101];
$array2 = [200846369 => 'Jan', 'name' => 'Jan', 'Olsen'.
'COMP518' => 69.,
'COMP519' => 52]);;
```

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

```php
array_count($array1, mode)
```

Arrays Basics

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

```php
$array = ['Peter', 3, 'a' => 101];
$array2 = [200846369 => 'Jan', 'name' => 'Jan', 'Olsen'.
'COMP518' => 69.,
'COMP519' => 52]);;
```

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

```php
array_count($array1, mode)
```
Arrays

- It is possible to omit the keys when using the `array` construct:
  
  ```php
  $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($array)`
  ~ returns a natural number-indexed array containing the keys of `$array`

- All the values of an array can be retrieved using `array_values($array)`
  ~ returns a natural number-indexed array containing the values stored in `$array`

Arrays Basics

- PHP allows the construct: `unset($array[$key])` removes a key-value pair from an array

Arrays: foreach-loop

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

  **Example 1:**
  ```php
  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:**
  ```php
  $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 1 to Peter
  The array maps 0 to Peter

Arrays Foreach-loops

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

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

  ```php
  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-loops

- **Example 3:**
  ```php
  $arr6 = array("name" => "Peter", "year" => 2009);
  foreach ($arr6 as $key => $value) { print "The array maps $key to $value\n";
  } unset($value = ",\"modified\"; // Changing $value
  }
  ```

  The array maps Peter to 2009
  The array maps name to 2009

Arrays Foreach-loops

- **Example 4:**
  ```php
  foreach ($arr6 as $key => $value) print "The array maps $key to $value\n";
  ```

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

Arrays Foreach-loops

- `$array[] = value` allows the construct:

  ```php
  $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

  ```php
  $arr4 = [1, 2, 3];
  $arr4[] = 5; // 0 => 5
  $arr4[] = 42; // 1 => 42
  $arr4[] = 33; // 2 => 33
  ```

Arrays: foreach-loop

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

  ```php
  $arr = array("name" => "Peter", "year" => 2009);
  unset($arr["name"]); // Removes the name 'Peter'
  unset($arr[1]); // Removes the whole array
  ```

Arrays: foreach-loop

- **Example:**
  ```php
  foreach ($arr6 as $key => $value) print "The array maps $key to $value\n";
  ```

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

Arrays: foreach-loop

- **Example:**
  ```php
  foreach ($arr6 as $key => $value) print "The array maps $key to $value\n";
  ```

  The array maps name to Peter
  The array maps year to 2009
Arrays Foreach-loops

Array Assignments

• In JavaScript, arrays were objects and as a consequence, array assignments were done by reference.

• In PHP, this is not the case.

```php
$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";
$array2[1] += 10000;
echo "$array2[1] = $array1[1],\n";
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
(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.

```php
$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] = 10002 | $array2[1] = 10002\n";
```

(2) 24 more bytes
(3) 0 more bytes

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)`
  shifts 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


• Read
  • Language Reference: Types: Arrays
  • Language Reference: Control Structures: foreach