Search

Keyboard Shortcuts

?
: This help

j
: Next menu item

k
: Previous menu item

g p
: Previous man page

g n
: Next man page

G
: Scroll to bottom

g g
: Scroll to top

g h
: Goto homepage

g s
: Goto search
(current page)

/
: Focus search box

Управление ошибками »
« Побитовые операторы

- Руководство по PHP
- Справочник языка
- Операторы

Change language: Russian

Операторы сравнения

Операторы сравнения, как это видно из их названия, разрешают сравнивать между собой два значения. Могут также оказаться интересными для знакомства таблицы сравнения типов, поскольку в них показаны примеры сравнений, связанных с разными типами.

Операторы сравнения

Пример	Название	Результат
$a == $b	Равно	Возвращается **true**, если значение переменной *$a* после преобразования типов равно значению переменной *$b*.
$a === $b	Тождественно равно	Возвращается **true**, если значение переменной *$a* равно значению переменной *$b* и имеет тот же тип.
$a != $b	Не равно	Возвращается **true**, если значение переменной *$a* после преобразования типов не равно значению переменной *$b*.
$a <> $b	Не равно	Возвращается **true**, если значение переменной *$a* после преобразования типов не равно значению переменной *$b*.
$a !== $b	Тождественно не равно	Возвращается **true**, если значение переменной *$a* не равно значению переменной *$b* или они разных типов.
$a < $b	Меньше	Возвращается **true**, если значение переменной *$a* строго меньше значения переменной *$b*.
$a > $b	Больше	Возвращается **true**, если значение переменной *$a* строго больше значения переменной *$b*.
$a <= $b	Меньше или равно	Возвращается **true**, если значение переменной *$a* меньше или равно значению переменной *$b*.
$a >= $b	Больше или равно	Возвращается **true**, если значение переменной *$a* больше или равно значению переменной *$b*.
$a <=> $b	Космический корабль (spaceship)	Целое число (int) меньше, больше или равное нулю, когда значение переменной *$a* меньше, больше или равно значению переменной *$b*.

Если оба операнда — строки, содержащие числа, или один операнд — число, а другой — строка, содержащая числа, то сравнение выполняется численно. Эти правила также справедливы для оператора switch. Тип не преобразовывается при сравнениях вида `===` или `!==`, поскольку это включает сравнение типа, а также значения.

Внимание

До PHP 8.0.0, если строка (string) сравнивалась с числом или строкой, содержащей число, то строка (string) преобразовывалась в число перед выполнением сравнения. Это могло привести к неожиданным результатам, что можно увидеть на следующем примере:

```
<?php

var_dump(0 == "a");
var_dump("1" == "01");
var_dump("10" == "1e1");
var_dump(100 == "1e2");

switch ("a") {
case 0:
echo "0";
break;
case "a":
echo "a";
break;
}
```

Результат выполнения приведённого примера в PHP 7:

```
bool(true)
bool(true)
bool(true)
```

```
bool(true)
0
```

Результат выполнения приведённого примера в PHP 8:

```
bool(false)
bool(true)
bool(true)
bool(true)
a
```

```php
<?php
// Целые числа
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1

// Числа с плавающей точкой
echo 1.5 <=> 1.5; // 0
echo 1.5 <=> 2.5; // -1
echo 2.5 <=> 1.5; // 1

// Строки
echo "a" <=> "a"; // 0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1
echo "a" <=> "aa"; // -1
echo "zz" <=> "aa"; // 1

// Массивы
echo [] <=> []; // 0
echo [1, 2, 3] <=> [1, 2, 3]; // 0
echo [1, 2, 3] <=> []; // 1
echo [1, 2, 3] <=> [1, 2, 1]; // 1
echo [1, 2, 3] <=> [1, 2, 4]; // -1

// Объекты
$a = (object) ["a" => "b"];
$b = (object) ["a" => "b"];
echo $a <=> $b; // 0
$a = (object) ["a" => "b"];
$b = (object) ["a" => "c"];
echo $a <=> $b; // -1
$a = (object) ["a" => "c"];
$b = (object) ["a" => "b"];
echo $a <=> $b; // 1

// сравниваются не только значения; ключи также должны совпадать
$a = (object) ["a" => "b"];
$b = (object) ["b" => "b"];
echo $a <=> $b; // 1
?>
```

Для различных типов сравнение происходит в соответствии со следующей таблицей (по порядку).

Сравнение типов

Тип операнда 1	Тип операнда 2	Результат
null или string	string	**`null`** преобразуется в пустую строку (""), числовое или лексическое сравнение
bool или null	что угодно	Преобразуется в bool, **`false`** < **`true`**
object	object	Встроенные классы могут определять свои правила сравнения, объекты разных классов не сравниваются, про сравнение объектов одного класса рассказано в разделе «Сравнение

		объекта»
string, resource, int или float	string, resource, int или float	Строки и ресурсы переводятся в числа, обычная математика
array	array	Массив с меньшим числом элементов меньше, если ключ из первого массива не найден во втором массиве — массивы не могут сравниваться, иначе идёт сравнение значений (смотрите пример ниже)
array	что угодно	Тип array всегда больше
object	что угодно	Тип object всегда больше

Пример #1 Сравнение boolean/null

```
<?php

// Логические значения и null всегда сравниваются как логические
var_dump(1 == TRUE); // TRUE — то же, что и (bool)1 == TRUE
var_dump(0 == FALSE); // TRUE — то же, что и (bool)0 == FALSE
var_dump(100 < TRUE); // FALSE — то же, что и (bool)100 < TRUE
var_dump(-10 < FALSE); // FALSE — то же, что и (bool)-10 < FALSE
var_dump(min(-100, -10, NULL, 10, 100)); // NULL — (bool)NULL < (bool)-100 это FALSE < TRUE
```

Пример #2 Алгоритм сравнения обычных массивов

```
<?php

// Массивы сравниваются как в этом примере — со стандартными операторами сравнения, а также оператором «космический
корабль» (spaceship).
function standard_array_compare($op1, $op2)
{
if (count($op1) < count($op2)) {
return -1; // $op1 < $op2
} elseif (count($op1) > count($op2)) {
return 1; // $op1 > $op2
}

foreach ($op1 as $key => $val) {
if (!array_key_exists($key, $op2)) {
return 1;
} elseif ($val < $op2[$key]) {
return -1;
} elseif ($val > $op2[$key]) {
return 1;
}
}

return 0; // $op1 == $op2
}
```

Внимание

Сравнение чисел с плавающей точкой

По причинам, связанным со способом внутреннего представления чисел с плавающей точкой (float), не нужно проверять два числа с плавающей точкой (float) на равенство.

Подробнее об этом можно узнать в документации по типу float.

> **Замечание**: Когда пишут код, помнят, что жонглирование типами в PHP не всегда даёт предсказуемый результат при сравнении значений разных типов, особенно при сравнении целых чисел (int) с логическими значениями (bool) или целых чисел (int) со строками (string). Поэтому лучше пользоваться операторами `===` и `!==`, а не `==` и `!=`.

Несравнимые значение

Хотя тождественные сравнения (`===` и `!==`) можно применять к произвольным значениям, другие операторы сравнения лучше применять только к сравнимым значениям. Результат сравнения несравнимых значений не определён и на него не нужно полагаться.

Смотрите также

- strcasecmp()
- strcmp()
- Операторы, работающие с массивами
- Типы

Тернарный оператор

Ещё один условный оператор — тернарный оператор «?:».

Пример #3 Присваивание значения по умолчанию

```php
<?php

// Пример выражения с тернарным оператором
$action = (empty($_POST['action'])) ? 'default' : $_POST['action'];

// Код выше аналогичен блоку с конструкциями if/else
if (empty($_POST['action'])) {
$action = 'default';
} else {
$action = $_POST['action'];
}
```

Выражение `(expr1) ? (expr2) : (expr3)` интерпретируется как expr2, если expr1 равно **`true`**, или как expr3, если expr1 равно **`false`**.

Можно не писать среднюю часть тернарного оператора. Выражение `expr1 ?: expr3` оценивается как результат выражения expr1, если оно оценивается как **`true`**, иначе как результат выражения expr3. Выражение expr1 оценивается только один раз.

> **Замечание**: Обратите внимание, что тернарный оператор — это выражение, и он оценивается не как переменная, а как результат выражения. Это важно, если нужно вернуть переменную по ссылке. Выражение `return $var == 42 ? $a : $b;` не будет работать в функции, возвращающей значение по ссылке, а в более поздних версиях PHP также будет выдано предупреждение.

> **Замечание**:
>
> Рекомендовано избегать «нагромождения» тернарных выражений. Поведение PHP при указании более чем одного тернарного оператора без скобок в одном выражении неочевидно в сравнении с другими языками. Впрямь, до PHP 8.0.0 троичные выражения оценивались как левоассоциативные, а не правоассоциативные, как в большей части других языков программирования. Опора на левую ассоциативность устарела начиная с PHP 7.4.0. Начиная с PHP 8.0.0 тернарный оператор неассоциативен.

> **Пример #4 Неочевидное поведение тернарного оператора**
>
> ```php
> <?php
>
> // кажется, что следующий код выведет «true»
> echo (true ? 'true' : false ? 't' : 'f');
>
> // однако он выводит «t» до PHP 8.0.0
> // это потому, что тернарные выражения левоассоциативны
>
> // следующая запись — более очевидная версия того же кода, который показан выше
> echo ((true ? 'true' : false) ? 't' : 'f');
>
> // здесь видно, что первое выражение оценивается как строковое «true», которое
> // оценивается как логическое (bool) true, поэтому возвращает истинную ветвь
> // второго тернарного выражения.
> ```

Замечание:

Цепочка коротких тернарных операторов (`?:`), однако, стабильна и ведёт себя обоснованно. Она будет оценивать первый аргумент, который оценивается как не ложное значение. Обратите внимание, что неопределённые значения все равно вызовут предупреждение.

Пример #5 Цепочка коротких тернарных операторов

```
<?php
echo 0 ?: 1 ?: 2 ?: 3, PHP_EOL; // 1
echo 0 ?: 0 ?: 2 ?: 3, PHP_EOL; // 2
echo 0 ?: 0 ?: 0 ?: 3, PHP_EOL; // 3
?>
```

Оператор объединения с null

Другой полезный сокращённый оператор — это оператор объединения с NULL — «??» (null coalescing).

Пример #6 Присваивание значения по умолчанию

```
<?php

// Пример работы с оператором нулевого слияния
$action = $_POST['action'] ?? 'default';

// Пример выше аналогичен этому выражению с if/else
if (isset($_POST['action'])) {
$action = $_POST['action'];
} else {
$action = 'default';
}
```

Выражение `(expr1) ?? (expr2)` вычисляется так: expr2, если expr1 равно **`null`**, иначе expr1.

Этот оператор не вызывает предупреждения или ошибки, если левый операнд не существует, точно как языковая конструкция isset(). Это очень полезно для ключей массива.

Замечание: Обратите внимание, оператор объединения с NULL — это выражение, и он оценивается не как переменная, а как результат вычисления выражения. Это важно, если нужно вернуть значение по ссылке. Выражение `return $foo ?? $bar;` в функции, возвращающей ссылку, будет не работать, а выводить предупреждение.

Замечание:

У оператора объединения с NULL низкий приоритет. То есть при смешивании его с другими операторами (например, с операторами конкатенации строк или арифметическими операторами), скорее всего, потребуются круглые скобки.

```
<?php

// Вызывает предупреждение о том, что $name не определено.
print 'Mr. ' . $name ?? 'Anonymous';

// Выведет "Mr. Anonymous"
print 'Mr. ' . ($name ?? 'Anonymous');
```

Замечание:

Обратите внимание, оператор объединения с NULL разрешает простую вложенность:

Пример #7 Вложенный оператор null coalescing

```
<?php

$foo = null;
$bar = null;
$baz = 1;
```

```
$qux = 2;

echo $foo ?? $bar ?? $baz ?? $qux; // выведет 1
```

+ add a note

User Contributed Notes 13 notes

up
down
173
crazy888s at hotmail dot com ¶
14 years ago
I couldn't find much info on stacking the new ternary operator, so I ran some tests:

```
<?php
echo 0 ?: 1 ?: 2 ?: 3; //1
echo 1 ?: 0 ?: 3 ?: 2; //1
echo 2 ?: 1 ?: 0 ?: 3; //2
echo 3 ?: 2 ?: 1 ?: 0; //3

echo 0 ?: 1 ?: 2 ?: 3; //1
echo 0 ?: 0 ?: 2 ?: 3; //2
echo 0 ?: 0 ?: 0 ?: 3; //3
?>
```

It works just as expected, returning the first non-false value within a group of expressions.
up
down
7
admin at zeros dot co dot id ¶
1 year ago
Please be careful when you try to compare strings that have a plus sign `+` at the beginning (such as phone number, etc). When you use the Equal operator `==` PHP will ignore the plus sign. Use Identical operator `===` instead

Example:

```
$str1 = "62";
$str2 = "+62";

var_dump($str1 == $str2); // bool(true)
var_dump($str1 === $str2); // bool(false)
```
up
down
22
adam at caucho dot com ¶
17 years ago
Note: according to the spec, PHP's comparison operators are not transitive. For example, the following are all true in PHP5:

```
"11" < "a" < 2 < "11"
```

As a result, the outcome of sorting an array depends on the order the elements appear in the pre-sort array. The following code will dump out two arrays with *different* orderings:

```
<?php
$a = array(2, "a", "11", 2);
$b = array(2, "11", "a", 2);
sort($a);
var_dump($a);
sort($b);
var_dump($b);
```

```
?>
```

This is not a bug report -- given the spec on this documentation page, what PHP does is "correct". But that may not be what was intended...

up
down
4

Sumon Mahmud ¶

4 years ago

Extending from here: https://www.php.net/manual/en/language.operators.comparison.php#121907

```
$a = ['a' => 1, 'b' => 2, 'c' => 3, 'e' => 4];
$b = ['a' => 1, 'b' => 2, 'd' => 3, 'e' => 4];

echo $a > $b; // 0
echo $b > $a; // 0
echo $a <$b; // 0
echo $b < $a; // 0
```

If using spaceship operator then it is returning true like :

```
echo $a <=> $b; //1
echo $b <=> $a; //1
echo $a <=> $b; //1
echo $b <=> $a; //1
```

up
down
1

Hayley Watson ¶

5 months ago

Between the "shortcut ternary" (aka "elvis") and "spaceship" operators, you can write some quite compact comparison functions for usort and its ilk.

If you want to sort an array of associative arrays by several different keys you can chain them in the same way that you can list column names in an SQL ORDER BY clause.

```
<?php
usort($array, fn($a, $b) => $a['a'] <=> $b['a']
?: $b['b'] <=> $a['b']
?: $a['c'] <=> $b['c']);
?>
```

Will sort the array by column 'a', then by column 'b' descending, then by column 'c'; or in SQL-speak 'ORDER BY a, b DESC, c".

up
down
17

rshawiii at yahoo dot com ¶

18 years ago

You can't just compare two arrays with the === operator
like you would think to find out if they are equal or not. This is more complicated when you have multi-dimensional arrays. Here is a recursive comparison function.

```
<?php
/**
* Compares two arrays to see if they contain the same values. Returns TRUE or FALSE.
* usefull for determining if a record or block of data was modified (perhaps by user input)
* prior to setting a "date_last_updated" or skipping updating the db in the case of no change.
*
* @param array $a1
* @param array $a2
* @return boolean
*/
```

```
function array_compare_recursive($a1, $a2)
{
if (!(is_array($a1) and (is_array($a2)))) { return FALSE;}

if (!count($a1) == count($a2))
{
return FALSE; // arrays don't have same number of entries
}

foreach ($a1 as $key => $val)
{
if (!array_key_exists($key, $a2))
{return FALSE; // uncomparable array keys don't match
}
elseif (is_array($val) and is_array($a2[$key])) // if both entries are arrays then compare recursive
{if (!array_compare_recursive($val,$a2[$key])) return FALSE;
}
elseif (!($val === $a2[$key])) // compare entries must be of same type.
{return FALSE;
}
}
return TRUE; // $a1 === $a2
}
?>
```

up
down
14

bishop ¶

17 years ago

When you want to know if two arrays contain the same values, regardless of the values' order, you cannot use "==" or "===". In other words:

```
<?php
(array(1,2) == array(2,1)) === false;
?>
```

To answer that question, use:

```
<?php
function array_equal($a, $b) {
return (is_array($a) && is_array($b) && array_diff($a, $b) === array_diff($b, $a));
}
?>
```

A related, but more strict problem, is if you need to ensure that two arrays contain the same key=>value pairs, regardless of the order of the pairs. In that case, use:

```
<?php
function array_identical($a, $b) {
return (is_array($a) && is_array($b) && array_diff_assoc($a, $b) === array_diff_assoc($b, $a));
}
?>
```

Example:

```
<?php
$a = array (2, 1);
$b = array (1, 2);
// true === array_equal($a, $b);
// false === array_identical($a, $b);

$a = array ('a' => 2, 'b' => 1);
$b = array ('b' => 1, 'a' => 2);
```

```
// true === array_identical($a, $b)
// true === array_equal($a, $b)
?>

(See also the solution "rshawiii at yahoo dot com" posted)
```

up
down
2

gfilippakis at sleed dot gr ¶

10 months ago

```
Please note that using the null coalescing operator to check properties on a class that has the __get magic method
(without an __isset magic method) invokes the magic method.

For example:

<?php

class A
{
public function __get($property)
{
echo 'Called __get for ' . $property . PHP_EOL;
}
}

$a = new A();

echo 'Trying null coalescing operator' . PHP_EOL;
$b = $a->test ?? 5;

echo 'Trying isset()' . PHP_EOL;
if (isset($a->test)) {
$b = $a->test;
} else {
$b = 5;
}

?>
```

up
down
6

Tahazzot ¶

2 years ago

```
Very careful when reading PHP documentation, Here's a lot of miss information.

According to documentation, They say's (int) 0 == (string) "a" is true. But it is not in PHP 8.

var_dump(0 == "a"); // 0 == 0 -> true

Now In PHP 8 it's False.
```

up
down
4

Marcin Kuzawiski ¶

8 years ago

```
A < B and still B < A...

$A = [1 => 1, 2 => 0, 3 => 1];
$B = [1 => 1, 3 => 0, 2 => 1];

var_dump($A < $B); // TRUE
var_dump($B < $A); // TRUE
```

```
var_dump($A > $B); // TRUE
var_dump($B > $A); // TRUE

Next - C and D are comparable, but neither C < D nor D < C (and still C != D)...

$C = [1 => 1, 2 => 1, 3 => 0];
$D = [1 => 1, 3 => 1, 2 => 0];

var_dump($C < $D); // FALSE
var_dump($D < $C); // FALSE

var_dump($C > $D); // FALSE
var_dump($D > $C); // FALSE

var_dump($D == $C); // FALSE
```

up
down
3

niall at maranelda dot org ¶

6 years ago

```
Care must be taken when using the spaceship operator with arrays that do not have the same keys:

- Contrary to the notes above ("Example #2 Transcription of standard array comparison"), it does *not* return null if the left-hand array contains a key that the right-hand array does not.
- Because of this, the result depends on the order you do the comparison in.

For example:

<?php
$a = ['a' => 1, 'b' => 2, 'c' => 3, 'e' => 4];
$b = ['a' => 1, 'b' => 2, 'd' => 3, 'e' => 4];

var_dump($a <=> $b); // int(1) : $a > $b because $a has the 'c' key and $b doesn't.

var_dump($b <=> $a); // int(1) : $b > $a because $b has the 'd' key and $a doesn't.
?>
```

up
down
4

Cuong Huy To ¶

12 years ago

```
In the table "Comparison with Various Types", please move the last line about "Object" to be above the line about "Array", since Object is considered to be greater than Array (tested on 5.3.3)

(Please remove my "Anonymous" post of the same content before. You could check IP to see that I forgot to type my name)
```

up
down
0

Ryan Mott ¶

4 years ago

```
Searching for "double question mark" operator should find this page (and hopefully after this comment the crawlers will agree)
```

＋add a note

- Операторы
  - Приоритет
  - Арифметика
  - Инкремент и декремент
  - Присваивание
  - Побитовые операторы
  - Сравнение