- Garbage Collection
- DTrace Dynamic Tracing

Function Reference
- Affecting PHP's Behaviour
- Audio Formats Manipulation
- Authentication Services
- Command Line Specific Extensions
- Compression and Archive Extensions
- Cryptography Extensions
- Database Extensions
- Date and Time Related Extensions
- File System Related Extensions
- Human Language and Character Encoding Support
- Image Processing and Generation
- Mail Related Extensions
- Mathematical Extensions
- Non-Text MIME Output
- Process Control Extensions
- Other Basic Extensions
- Other Services
- Search Engine Extensions
- Server Specific Extensions
- Session Extensions
- Text Processing
- Variable and Type Related Extensions
- Web Services
- Windows Only Extensions
- XML Manipulation
- GUI Extensions

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

Целые числа »

« NULL

Change language: Russian

Логические значения

У логического типа (bool) есть только два значения и они выражают истинность значения. Он может быть либо **`true`**, либо **`false`**.

Синтаксис

Чтобы задать логический литерал bool, указывают константы **`true`** или **`false`**. Они обе регистронезависимы.

```
<?php

$foo = True; // Присвоить переменной $foo значение TRUE

?>
```

Обычно, некоторый оператор возвращает логическое значение bool, которое потом передаётся управляющей конструкции.

```
<?php

// == это оператор, который проверяет
// эквивалентность и возвращает boolean
if ($action == "show_version") {
echo "Версия 1.23";
}

// это необязательно...
if ($show_separators == TRUE) {
echo "<hr>\n";
}

// ...потому что следующее имеет тот же самый смысл:
if ($show_separators) {
echo "<hr>\n";
}

?>
```

Преобразование в логический тип

Чтобы явно преобразовать значение в логическое bool, пользуются приведением `(bool)`. Обычно это не нужно, поскольку значение в логическом контексте автоматически интерпретируется как значение логического типа (bool). Дополнительную информацию даёт раздел «Манипуляции с типами».

При преобразовании в логическое значение bool, следующие значения рассматриваются как **`false`**:

- само значение boolean **`false`**
- integer `0` (ноль)
- float `0.0` (ноль) и `-0.0` (минус ноль)
- пустая строка `""` и строка `"0"`
- массив без элементов
- особый тип NULL (включая неустановленные переменные)
- внутренние объекты, которые перегружают своё поведение приведения к логическому типу. Например: объекты SimpleXML, созданные из пустых элементов без атрибутов.

Все остальные значения считаются **`true`** (включая resource и **`NAN`**).

Внимание

Число `-1` рассматривается как **`true`**, как и любое другое ненулевое (отрицательное или положительное) число!

```
<?php
```

```
var_dump((bool) ""); // bool(false)
var_dump((bool) "0"); // bool(false)
var_dump((bool) 1); // bool(true)
var_dump((bool) -2); // bool(true)
var_dump((bool) "foo"); // bool(true)
var_dump((bool) 2.3e5); // bool(true)
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array()); // bool(false)
var_dump((bool) "false"); // bool(true)

?>
```

+add a note

User Contributed Notes 11 notes

up
down
967
Fred Koschara ¶
10 years ago

```
Ah, yes, booleans - bit values that are either set (TRUE) or not set (FALSE). Now that we have 64 bit compilers using an int variable for booleans, there is *one* value which is FALSE (zero) and 2**64-1 values that are TRUE (everything else). It appears there's a lot more truth in this universe, but false can trump anything that's true...

PHP's handling of strings as booleans is *almost* correct - an empty string is FALSE, and a non-empty string is TRUE - with one exception: A string containing a single zero is considered FALSE. Why? If *any* non-empty strings are going to be considered FALSE, why *only* a single zero? Why not "FALSE" (preferably case insensitive), or "0.0" (with how many decimal places), or "NO" (again, case insensitive), or ... ?

The *correct* design would have been that *any* non-empty string is TRUE - period, end of story. Instead, there's another GOTCHA for the less-than-completely-experienced programmer to watch out for, and fixing the language's design error at this late date would undoubtedly break so many things that the correction is completely out of the question.

Speaking of GOTCHAs, consider this code sequence:
<?php
$x=TRUE;
$y=FALSE;
$z=$y OR $x;
?>

Is $z TRUE or FALSE?

In this case, $z will be FALSE because the above code is equivalent to <?php ($z=$y) OR $x ?> rather than <?php $z=($y OR $x) ?> as might be expected - because the OR operator has lower precedence than assignment operators.

On the other hand, after this code sequence:
<?php
$x=TRUE;
$y=FALSE;
$z=$y || $x;
?>

$z will be TRUE, as expected, because the || operator has higher precedence than assignment: The code is equivalent to $z=($y OR $x).

This is why you should NEVER use the OR operator without explicit parentheses around the expression where it is being used.
```

up
down
153
Mark Simon ¶

6 years ago

```
Note for JavaScript developers:

In PHP, an empty array evaluates to false, while in JavaScript an empty array evaluates to true.

In PHP, you can test an empty array as <?php if(!$stuff) …; ?> which won’t work in JavaScript where you need to test the array length.

This is because in JavaScript, an array is an object, and, while it may not have any elements, it is still regarded as something.

Just a trap for young players who routinely work in both langauges.
```

up
down
83
goran77 at fastmail dot fm ¶
7 years ago

```
Just something that will probably save time for many new developers: beware of interpreting FALSE and TRUE as integers.
For example, a small function for deleting elements of an array may give unexpected results if you are not fully aware of what happens:

<?php

function remove_element($element, $array)
{
//array_search returns index of element, and FALSE if nothing is found
$index = array_search($element, $array);
unset ($array[$index]);
return $array;
}

// this will remove element 'A'
$array = ['A', 'B', 'C'];
$array = remove_element('A', $array);

//but any non-existent element will also remove 'A'!
$array = ['A', 'B', 'C'];
$array = remove_element('X', $array);
?>

The problem here is, although array_search returns boolean false when it doesn't find specific element, it is interpreted as zero when used as array index.

So you have to explicitly check for FALSE, otherwise you'll probably loose some elements:

<?php
//correct
function remove_element($element, $array)
{
$index = array_search($element, $array);
if ($index !== FALSE)
{
unset ($array[$index]);
}
return $array;
}
```

up
down
20
asma dot gi dot 14 at gmail dot com ¶
2 years ago

```
Please keep in mind that the result of 0 == 'whatever' is true in PHP Version 7 and false in PHP version 8.
```

up
down
57
terminatorul at gmail dot com ¶
16 years ago
```
Beware that "0.00" converts to boolean TRUE !

You may get such a string from your database, if you have columns of type DECIMAL or CURRENCY. In such cases you have to explicitly check if the value is != 0 or to explicitly convert the value to int also, not only to boolean.
```
up
down
6
asma dot gi dot 14 at gmail dot com ¶
2 years ago
```
when using echo false; or print false; the display will be empty but when using echo 0; or print 0; the display will be 0.
```
up
down
44
Steve ¶
16 years ago
```
PHP does not break any rules with the values of true and false. The value false is not a constant for the number 0, it is a boolean value that indicates false. The value true is also not a constant for 1, it is a special boolean value that indicates true. It just happens to cast to integer 1 when you print it or use it in an expression, but it's not the same as a constant for the integer value 1 and you shouldn't use it as one. Notice what it says at the top of the page:

A boolean expresses a truth value.

It does not say "a boolean expresses a 0 or 1".

It's true that symbolic constants are specifically designed to always and only reference their constant value. But booleans are not symbolic constants, they are values. If you're trying to add 2 boolean values you might have other problems in your application.
```
up
down
18
Mark Simon ¶
6 years ago
```
Note on the OR operator.

A previous comment notes the trap you can fall into with this operator. This is about its usefulness.

Both OR and || are short-circuited operators, which means they will stop evaluating once they reach a TRUE value. By design, OR is evaluated after assignment (while || is evaluated before assignment).

This has the benefit of allowing some simple constructions such as:

<?php
$stuff=getStuff() or die('oops');
$thing=something() or $thing=whatever();
?>

The first example, often seen in PERL, could have been written as <?php if(!$stuff=getStuff()) die('oops'); ?> but reads a little more naturally. I have often used it in situations where null or false indicate failure.

The second allows for an alternative value if a falsy one is regarded as insufficient. The following example

<?php
$page=@$_GET['page'] or $page=@$_COOKIE['page'] or $page=1;
?>

is a simple way sequencing alternative values. (Note the usual warnings about using the @ operator or accepting unfiltered input …)
```

All this presupposes that 0 is also an unacceptable value in the situation.
up
down
35
Wackzingo ¶
16 years ago
It is correct that TRUE or FALSE should not be used as constants for the numbers 0 and 1. But there may be times when it might be helpful to see the value of the Boolean as a 1 or 0. Here's how to do it.

```
<?php
$var1 = TRUE;
$var2 = FALSE;

echo $var1; // Will display the number 1

echo $var2; //Will display nothing

/* To get it to display the number 0 for
a false value you have to typecast it: */

echo (int)$var2; //This will display the number 0 for false.
?>
```
up
down
35
artktec at gmail dot com ¶
16 years ago
Note you can also use the '!' to convert a number to a boolean, as if it was an explicit (bool) cast then NOT.

So you can do something like:

```
<?php
$t = !0; // This will === true;
$f = !1; // This will === false;
?>
```

And non-integers are casted as if to bool, then NOT.

Example:

```
<?php
$a = !array(); // This will === true;
$a = !array('a'); // This will === false;
$s = !""; // This will === true;
$s = !"hello"; // This will === false;
?>
```

To cast as if using a (bool) you can NOT the NOT with "!!" (double '!'), then you are casting to the correct (bool).

Example:

```
<?php
$a = !!array(); // This will === false; (as expected)
/*
This can be a substitute for count($array) > 0 or !(empty($array)) to check to see if an array is empty or not (you would use: !!$array).
*/

$status = (!!$array ? 'complete' : 'incomplete');

$s = !!"testing"; // This will === true; (as expected)
```

```
/*
Note: normal casting rules apply so a !!"0" would evaluate to an === false
*/
?>
```

up
down
6
marklgr ¶
8 years ago

```
For those wondering why the string "0" is falsy, consider that a good deal of input data is actually string-typed, even when it is semantically numeral.

PHP often tries to autoconvert these strings to numeral, as the programmer certainly intended (try 'echo "2"+3'). Consequently, PHP designers decided to treat 0 and "0" similarly, ie. falsy, for consistency and to avoid bugs where the programmer believes he got a true numeral that would happen to be truthy when zero.
```

+add a note