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

Строки »
« Исполнение

- Руководство по PHP
- Справочник языка
- Операторы

Change language: Russian

Логические операторы

Логические операторы

Пример	Название	Результат
$a and $b	И	**true**, если и *$a*, и *$b* **true**.
$a or $b	Или	**true**, если или *$a*, или *$b* **true**.
$a xor $b	Исключающее или	**true**, если *$a*, или *$b* **true**, но не оба.
! $a	Отрицание	**true**, если *$a* не **true**.
$a && $b	И	**true**, если и *$a*, и *$b* **true**.
$a || $b	Или	**true**, если или *$a*, или *$b* **true**.

Причина существования двух разных операторов «И» и «ИЛИ» в том, что они работают с разным приоритетом (смотрите таблицы в разделе «Приоритет оператора»).

Пример #1 Объяснение логических операторов

```php
<?php

// --------------------
// Функция foo() никогда не будет вызвана, т. к. эти операторы шунтирующие (short-circuit)

$a = (false && foo());
$b = (true || foo());
$c = (false and foo());
$d = (true or foo());

// --------------------
// У оператора «||» больший приоритет, чем у «or»

// Результат выражения (false || true) присваивается переменной $e
// Действует как: ($e = (false || true))
$e = false || true;

// Константа false присваивается переменной $f, а затем значение true игнорируется
// Действует как: (($f = false) or true)
$f = false or true;

var_dump($e, $f);

// --------------------
// У оператора «&&» больший приоритет, чем у «and»

// Результат выражения (true && false) присваивается переменной $g
// Действует как: ($g = (true && false))
$g = true && false;

// Константа true присваивается переменной $h, а затем значение false игнорируется
// Действует как: (($h = true) and false)
$h = true and false;

var_dump($g, $h);
```

Вывод приведённого примера будет похож на:

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

User Contributed Notes 14 notes

up
down
301
Lawrence ¶
16 years ago
Note that PHP's boolean operators *always* return a boolean value... as opposed to other languages that return the value of the last evaluated expression.

For example:

```
$a = 0 || 'avacado';
print "A: $a\n";
```

will print:

```
A: 1
```

in PHP -- as opposed to printing "A: avacado" as it would in a language like Perl or JavaScript.

This means you can't use the '||' operator to set a default value:

```
$a = $fruit || 'apple';
```

instead, you have to use the '?:' operator:

```
$a = ($fruit ? $fruit : 'apple');
```

up
down
63
dumitru at floringabriel dot com ¶
6 years ago
In addition to what Lawrence said about assigning a default value, one can now use the Null Coalescing Operator (PHP 7). Hence when we want to assign a default value we can write:

```
$a = ($fruit ?? 'apple');
//assigns the $fruit variable content to $a if the $fruit variable exists or has a value that is not NULL, or assigns the value 'apple' to $a if the $fruit variable doesn't exists or it contains the NULL value
```

up
down
27
thisleenobleNOSPAMPLEASE at mac dot com ¶
7 years ago
In order to kind of emulate the way javascript assigns the first non-false value in an expression such as this:

```
var v = a || b || c || d;
```

I wrote a little helper method that I put in a function dump library (here presented as a bare function):

```
<?php
function either($a, $b){
$val = $a ? $a : $b;
/*
Yes, I know the fixed parameters in the function
are redundant since I could just use func_get_args,
but in most instances I'll be using this a replacement
for the ternary operator and only passing two values.
I don't want to invoke the additional process below
unless I REALLY have to.
*/
$args = func_get_args();
```

```
if($val === false && count($args) > 2){
$args = array_slice($args, 2);

foreach($args as $arg){
if($arg !== false){
$val = $arg;
break;
}
}
}
return $val;
}
?>
```

Now instead of:

```
$v = $a ? $a : $b;
```

I write:

```
$v = either($a, $b);
```

but more importantly, instead of writing:

```
$v = $a ? $a : ($b ? $b : $c);
```

I write:

```
$v = either($a, $b, $c);
```

or indeed:

```
$v = either($a, $b, $c, $d, $e, $f, $g, $h);
```

up
down
2

martinholtcbi at gmail dot com ¶

1 year ago

In PHP, the || operator only ever returns a boolean. For a chainable assignment operator, use the ?: "Elvis" operator.

JavaScript:

```
let a = false;
let b = false;
let c = true;
let d = false;
let e = a || b || c || d;
// e === c
```

```
<?PHP
$a = false;
$b = false;
$c = true;
$d = false;
$e = $a ?: $b ?: $c ?: $d;
// $e === $c
?>
```

Credit to @egst and others for the insight. This is merely a rewording for (formerly) lost JavaScript devs like myself.

up
down
57

pepesantillan at gmail dot com ¶

16 years ago

worth reading for people learning about php and programming: (adding extras <?php ?> to get highlighted code)

about the following example in this page manual:
Example#1 Logical operators illustrated

...

```
<?php
// "||" has a greater precedence than "or"
$e = false || true; // $e will be assigned to (false || true) which is true
$f = false or true; // $f will be assigned to false
var_dump($e, $f);

// "&&" has a greater precedence than "and"
$g = true && false; // $g will be assigned to (true && false) which is false
$h = true and false; // $h will be assigned to true
var_dump($g, $h);
?>
```

_____end of my quote...

If necessary, I wanted to give further explanation on this and say that when we write:
`$f = false or true; // $f will be assigned to false`
the explanation:

"||" has a greater precedence than "or"

its true. But a more acurate one would be

"||" has greater precedence than "or" and than "=", whereas "or" doesnt have greater precedence than "=", so

```
<?php
$f = false or true;

//is like writting

($f = false ) or true;

//and

$e = false || true;

is the same as

$e = (false || true);

?>
```

same goes for "&&" and "AND".

If you find it hard to remember operators precedence you can always use parenthesys - "(" and ")". And even if you get to learn it remember that being a good programmer is not showing you can do code with fewer words. The point of being a good programmer is writting code that is easy to understand (comment your code when necessary!), easy to maintain and with high efficiency, among other things.

up
down
13

egst ¶

2 years ago

In response to Lawrence about || always returning a boolean:

Instead of

```
$x ? $x : 'fallback'

you can also use the "elvis operator":

$x ?: 'fallback'

which is useful in cases, where the left-hand side of the ternary operator is too long type, is too complex to calculate
twice, or has side-effects.

It also combines nicely with the ?? operator, which is equivalent to an empty() check (both isset() and `!= false`):

$x->y ?? null ?: 'fallback';

instead of:

empty($x->y) ? $x->y : 'fallback'
```

up
down
20

momrom at freenet dot de ¶

14 years ago

```
Evaluation of logical expressions is stopped as soon as the result is known.
If you don't want this, you can replace the and-operator by min() and the or-operator by max().

<?php
function a($x) { echo 'Expression '; return $x; }
function b($x) { echo 'is '; return $x; }
function c($x) { echo $x ? 'true.' : 'false.' ;}

c( a( false ) and b( true ) ); // Output: Expression false.
c( min( a( false ), b( true ) ) ); // Output: Expression is false.

c( a( true ) or b( true ) ); // Output: Expression true.
c( max( a( true ), b( true ) ) ); // Output: Expression is true.
?>

This way, values aren't automaticaly converted to boolean like it would be done when using and or or. Therefore, if you
aren't sure the values are already boolean, you have to convert them 'by hand':

<?php
c( min( (bool) a( false ), (bool) b( true ) ) );
?>
```

up
down
21

phpnet at zc dot webhop dot net ¶

11 years ago

```
This works similar to javascripts short-curcuit assignments and setting defaults. (e.g. var a = getParm() || 'a default';)

<?php

($a = $_GET['var']) || ($a = 'a default');

?>

$a gets assigned $_GET['var'] if there's anything in it or it will fallback to 'a default'
Parentheses are required, otherwise you'll end up with $a being a boolean.
```

up
down
14

Andrew ¶

16 years ago

```
> <?php
> your_function() or return "whatever";
> ?>
```

doesn't work because return is not an expression, it's a statement. if return was a function it'd work fine. :/

up
down
6

anatoliy at ukhvanovy dot name ¶

9 years ago

If you want to use the '||' operator to set a default value, like this:

```
<?php
$a = $fruit || 'apple'; //if $fruit evaluates to FALSE, then $a will be set to TRUE (because (bool)'apple' == TRUE)
?>
```

instead, you have to use the '?:' operator:

```
<?php
$a = ($fruit ? $fruit : 'apple');//if $fruit evaluates to FALSE, then $a will be set to 'apple'
?>
```

But $fruit will be evaluated twice, which is not desirable. For example fruit() will be called twice:

```
<?php
function fruit($confirm) {
if($confirm)
return 'banana';
}
$a = (fruit(1) ? fruit(1) : 'apple');//fruit() will be called twice!
?>
```

But since «since PHP 5.3, it is possible to leave out the middle part of the ternary operator» (http://www.php.net/manual/en/language.operators.comparison.php#language.operators.comparison.ternary), now you can code like this:

```
<?php
$a = ($fruit ? : 'apple'); //this will evaluate $fruit only once, and if it evaluates to FALSE, then $a will be set to
'apple'
?>
```

But remember that a non-empty string '0' evaluates to FALSE!

```
<?php
$fruit = '1';
$a = ($fruit ? : 'apple'); //this line will set $a to '1'
$fruit = '0';
$a = ($fruit ? : 'apple'); //this line will set $a to 'apple', not '0'!
?>
```

up
down
10

peter dot kutak at NOSPAM dot gmail dot com ¶

16 years ago

```
$test = true and false; ---> $test === true
$test = (true and false); ---> $test === false
$test = true && false; ---> $test === false
```

NOTE: this is due to the first line actually being

```
($test = true) and false;
```

due to "&&" having a higher precedence than "=" while "and" has a lower one

up
down
3
void at informance dot info ¶
9 years ago
To assign default value in variable assignation, the simpliest solution to me is:

```
<?php
$v = my_function() or $v = "default";
?>
```

It works because, first, $v is assigned the return value from my_function(), then this value is evaluated as a part of a logical operation:
* if the left side is false, null, 0, or an empty string, the right side must be evaluated and, again, because 'or' has low precedence, $v is assigned the string "default"
* if the left side is none of the previously mentioned values, the logical operation ends and $v keeps the return value from my_function()

This is almost the same as the solution from [phpnet at zc dot webhop dot net], except that his solution (parenthesis and double pipe) doesn't take advantage of the "or" low precedence.

NOTE: "" (the empty string) is evaluated as a FALSE logical operand, so make sure that the empty string is not an acceptable value from my_function(). If you need to consider the empty string as an acceptable return value, you must go the classical "if" way.
up
down
4
brian at zickzickzick dot com ¶
10 years ago
This has been mentioned before, but just in case you missed it:

```
<?php
// Defaults --

//If you're trying to gat 'Jack' from:
$jack = false or 'Jack';

// Try:
$jack = false or $jack = 'Jack';

//The other option is:
$jack = false ? false : 'Jack';
?>
```
up
down
3
samantha at adrichem dot nu ¶
8 years ago
```
<?php
$res |= true;
var_dump($res);
?>
```

does not/no longer returns a boolean (php 5.6) instead it returns int 0 or 1
＋add a note

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