



- [Downloads](#)
- [Documentation](#)
- [Get Involved](#)
- [Help](#)
- 

[Dutch PHP Conference 2024](#)

[Getting Started](#)

[Introduction](#)

[A simple tutorial](#)

[Language Reference](#)

[Basic syntax](#)

[Types](#)

[Variables](#)

[Constants](#)

[Expressions](#)

[Operators](#)

[Control Structures](#)

[Functions](#)

[Classes and Objects](#)

[Namespaces](#)

[Enumerations](#)

[Errors](#)

[Exceptions](#)

[Fibers](#)

[Generators](#)

[Attributes](#)

[References Explained](#)

[Predefined Variables](#)

[Predefined Exceptions](#)

[Predefined Interfaces and Classes](#)

[Predefined Attributes](#)

[Context options and parameters](#)

[Supported Protocols and Wrappers](#)

[Security](#)

[Introduction](#)

[General considerations](#)

[Installed as CGI binary](#)

[Installed as an Apache module](#)

[Session Security](#)

[Filesystem Security](#)

[Database Security](#)

[Error Reporting](#)

[User Submitted Data](#)

[Hiding PHP](#)

[Keeping Current](#)

[Features](#)

[HTTP authentication with PHP](#)

[Cookies](#)

[Sessions](#)

[Dealing with XForms](#)

[Handling file uploads](#)

[Using remote files](#)

[Connection handling](#)

[Persistent Database Connections](#)

[Command line usage](#)

[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

[Арифметика »](#)

[« Операторы](#)

- [Руководство по PHP](#)
- [Справочник языка](#)
- [Операторы](#)

Change language: Russian

Приоритет оператора

Приоритет оператора определяет, насколько «тесно» он связывает между собой два выражения. Например, выражение `1 + 5 * 3` вычисляется как 16, а не 18, поскольку оператор умножения («*») имеет более высокий приоритет, чем оператор сложения («+»). Круглые скобки можно указывать для изменения порядка выполнения операторов. Например, выражение `(1 + 5) * 3` вычисляется как 18.

Если операторы имеют равный приоритет, то будут ли они выполняться справа налево или слева направо — зависит от их ассоциативности. Например, «-» — левоассоциативный оператор. Поэтому, выражение `1 - 2 - 3` сгруппируется как `(1 - 2) - 3` и пересчитается в -4. При этом оператор «=» — правоассоциативный, так что выражение `$a = $b = $c` сгруппируется как `$a = ($b = $c)`.

Неассоциативные операторы с одинаковым приоритетом нельзя указывать совместно. Например, выражение `1 < 2 > 1` не будет работать в PHP. При этом выражение `1 <= 1 == 1` будет, поскольку у оператора `==` более низкий приоритет, чем у оператора `<=`.

Ассоциативность относится только для двоичных (и тернарных) операторов. Унарные операторы бывают префиксными или постфиксными, поэтому это понятие к ним не относится. Например, `!!$a` можно сгруппировать только как `!(!$a)`.

Указание скобок, даже если это не строго необходимо, часто может улучшить читаемость кода за счёт явной группировки, а не опоры на приоритеты и ассоциативность.

В следующей таблице приведён список операторов, отсортированный по убыванию их приоритетов. Операторы, перечисленные в одной строке, имеют одинаковый приоритет, тогда порядок их выполнения будет определён тем, как они сгруппированы.

Порядок выполнения операторов		
Ассоциативность	Оператор	Дополнительная информация
(н/а)	<code>clone new</code>	clone и new
правая	<code>**</code>	арифметические операторы
(н/а)	<code>+ - ++ -- ~ (int) (float) (string) (array) (object) (bool) @</code>	арифметические операторы (унарные <code>+</code> и <code>-</code>), инкремент/декремент , побитовые операторы , приведение типов и оператор управления ошибками
левая	<code>instanceof</code>	типы
(н/а)	<code>!</code>	логические операторы
левая	<code>* / %</code>	арифметические операторы
левая	<code>+ - .</code>	арифметические операторы (бинарные <code>+</code> и <code>-</code>), операторы, работающие с массивами и строковые операторы (<code>.</code> до PHP 8.0.0)
левая	<code><< >></code>	побитовые операторы
левая	<code>.</code>	строковые операторы (начиная с PHP 8.0.0)
неассоциативна	<code>< <= > >=</code>	операторы сравнения
неассоциативна	<code>== != === !== <> <=></code>	операторы сравнения
левая	<code>&</code>	побитовые операторы и ссылки
левая	<code>^</code>	побитовые операторы
левая	<code> </code>	побитовые операторы
левая	<code>&&</code>	логические операторы
левая	<code> </code>	логические операторы
правая	<code>??</code>	операторы сравнения с null
неассоциативна	<code>? :</code>	тернарный оператор (левоассоциативный до PHP 8.0.0)
правая	<code>= += -= *= **= /= .= %= &= = ^= <<= >>= ??=</code>	операторы присваивания
(н/а)	<code>yield from</code>	yield from
(н/а)	<code>yield</code>	yield
(н/а)	<code>print</code>	print
левая	<code>and</code>	логические операторы

левая	xor	логические операторы
левая	or	логические операторы

Пример #1 Ассоциативность

```
<?php

$a = 3 * 3 % 5; // (3 * 3) % 5 = 4
// ассоциативность тернарных операторов отличается от C/C++
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2 (до PHP 8.0.0)

$a = 1;
$b = 2;
$a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5
```

Приоритет и ассоциативность оператора определяет только то, как группируется выражение, а не порядок его вычисления. Обычно PHP не указывает, в каком порядке вычисляются выражения, и нужно избегать кода, который предполагает специфический порядок вычисления, потому что поведение может меняться в разных версиях PHP или в зависимости от окружающего кода.

Пример #2 Неопределённый порядок вычисления

```
<?php

$a = 1;
echo $a + $a++; // может вывести как 2, так и 3

$i = 1;
$array[$i] = $i++; // может установить индекс как 1, так 2
```

Пример #3 +, - и . имеют одинаковый приоритет (до PHP 8.0.0)

```
<?php

$x = 4;
// следующий код может выдать неожиданный результат:
echo "x минус 1 равно " . $x-1 . ", ну, я надеюсь\n";

// поскольку он вычисляется таким образом (до PHP 8.0.0):
echo (("x минус один равно " . $x) - 1) . ", ну, я надеюсь\n";

// требуемый приоритет следует задать скобками:
echo "x минус 1 равно " . ($x-1) . ", ну, я надеюсь\n";
```

Результат выполнения приведённого примера:

```
-1, ну, я надеюсь
-1, ну, я надеюсь
x минус один равно 3, ну, я надеюсь
```

Замечание:

Хотя оператор = имеет более низкий приоритет, чем большая часть других операторов, PHP всё же разрешает делать так: `if (!$a = foo())`, в этом примере результат выполнения функции `foo()` будет присвоен переменной `$a`.

Список изменений

Версия	Описание
8.0.0	Объединение строк (.) теперь имеет более низкий приоритет, чем арифметическое сложение/вычитание (+ и -) и побитовый сдвиг влево/вправо (<< и >>); ранее он имел тот же приоритет, что и + и -, и более высокий приоритет, чем << и >>.
8.0.0	Тернарный оператор (? :) теперь неассоциативен; ранее он был левоассоциативным.
7.4.0	Практика, когда можно было опираться на приоритет конкатенации строк (.) при арифметических операциях сложения/вычитания (+ или -) или побитовом сдвиге влево/вправо (<< или >>), т. е. когда в

выражении без скобок они указаны вместе, не рекомендована.

7.4.0 Практика, когда можно было полагаться на левоассоциативность тернарного оператора (? :), т. е. вложение нескольких тернарных операторов без скобок, не рекомендована.

[+add a note](#)

User Contributed Notes 8 notes

[up](#)

[down](#)

213

[fabmlk ¶](#)

8 years ago

Watch out for the difference of priority between 'and vs &&' or '|| vs or':

```
<?php
$bool = true && false;
var_dump($bool); // false, that's expected
```

```
$bool = true and false;
var_dump($bool); // true, ouch!
?>
```

Because 'and/or' have lower priority than '=' but '||/&&' have higher.

[up](#)

[down](#)

39

[aaronw at catalyst dot net dot nz ¶](#)

6 years ago

If you've come here looking for a full list of PHP operators, take note that the table here is **not** complete. There are some additional operators (or operator-ish punctuation tokens) that are not included here, such as "->", "::", and "...".

For a really comprehensive list, take a look at the "List of Parser Tokens" page: <http://php.net/manual/en/tokens.php>

[up](#)

[down](#)

6

[sangala at seznam dot cz ¶](#)

1 year ago

Using cast and ternary operator can be unclear,
(Useful to know with: declare(strict_types = 1)).

```
<?php
$num_str="5";

$i1 = (int) isset($num_str) ? $num_str : 0;
$i2 = (int) (isset($num_str) ? $num_str : 0);
var_dump($i1);
var_dump($i2);
?>
```

Output:
string(1) "5"
int(5)

[up](#)

[down](#)

52

[Carsten Milkau ¶](#)

11 years ago

Beware the unusual order of bit-wise operators and comparison operators, this has often lead to bugs in my experience. For instance:

```
<?php if ( $flags & MASK == 1) do_something(); ?>
```

will not do what you might expect from other languages. Use

```
<?php if (($flags & MASK) == 1) do_something(); ?>
```

in PHP instead.

[up](#)

[down](#)

11

[ivan at dilber dot info ¶](#)

6 years ago

```
<?php
```

```
// Another tricky thing here is using && or || with ternary ?:
```

```
$x && $y ? $a : $b; // ($x && $y) ? $a : $b;
```

```
// while:
```

```
$x and $y ? $a : $b; // $x and ($y ? $a : $b);
```

```
?>
```

[up](#)

[down](#)

3

[tlili dot mokhtar at gmail dot com ¶](#)

2 years ago

An easy trick to get the result of the left shift operation (<<), e.g.

```
15 << 2 = 15 * (2*2) = 60
```

```
15 << 3 = 15 * (2*2*2) = 120
```

```
15 << 5 = 15 * (2*2*2*2*2) = 480
```

and so on...

So it's:

(number on left) multiplied by (number on right) times 2.

The same goes for the right shift operator (>>), where:

(number on left) divided by (number on right) times 2 e.g.

```
15 >> 2 = (15/2)/2 = 7/2 = 3 (use floor values if result is in decimals).
```

```
35 >> 3 = (((35/2)/2)/2) = (17/2)/2 = 8/2 = 4
```

[up](#)

[down](#)

1

[rvwoens at gmail dot com ¶](#)

1 year ago

Note that ?? has a low priority, so this can lead to unexpected results:

```
$a=[];
```

```
$a['aa']??'not set'
```

```
--> not set (as expected)
```

but

```
"lets see if it is set".$a['aa']??'not set'
```

```
--> notice; undefined index aa
```

```
--> lets see if it is set
```

so you need to use parenthesis

```
"lets see if it is set".($a['aa']??'not set')
```

[up](#)

[down](#)

1

[instatiendaweb at gmail dot com ¶](mailto:instatiendaweb@gmail.com)

2 years ago

```
//incorrect
```

```
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2
```

```
//Unparenthesized `a ? b : c ? d : e` is not supported. Use either `(a ? b : c) ? d : e` or `a ? b : (c ? d : e)`
```

```
//correct
```

```
$a = (true ? 0 : true) ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2
```

==> correction documentation.

[+add a note](#)

- [Операторы](#)
 - [Приоритет](#)
 - [Арифметика](#)
 - [Инкремент и декремент](#)
 - [Присваивание](#)
 - [Побитовые операторы](#)
 - [Сравнение](#)
 - [Управление ошибками](#)
 - [Исполнение](#)
 - [Логика](#)
 - [Строки](#)
 - [Массивы](#)
 - [Проверка типа](#)
- [Copyright © 2001-2024 The PHP Group](#)
- [My PHP.net](#)
- [Contact](#)
- [Other PHP.net sites](#)
- [Privacy policy](#)

