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

Les opérateurs arithmétiques »
« Les opérateurs

- Manuel PHP
- Référence du langage
- Les opérateurs

Change language: French

Submit a Pull Request Report a Bug

La priorité des opérateurs ¶

La priorité des opérateurs spécifie l'ordre dans lequel les valeurs doivent être analysées. Par exemple, dans l'expression `1 + 5 * 3`, le résultat est `16` et non `18`, car la multiplication ("*") a une priorité supérieure par rapport à l'addition ("+"). Des parenthèses peuvent être utilisées pour forcer la priorité, si nécessaire. Par exemple : `(1 + 5) * 3` donnera `18`.

Lorsque les opérateurs ont une priorité égale, leur association décide la façon dont les opérateurs sont groupés. Par exemple, "`-`" est une association par la gauche, ainsi `1 - 2 - 3` est groupé comme ceci `(1 - 2) - 3` et sera évalué à `-4`. D'un autre côté, "`=`" est une association par la droite, ainsi, `$a = $b = $c` est groupé comme ceci `$a = ($b = $c)`.

Les opérateurs, dont la priorité est égale, qui ne sont pas associatifs, ne peuvent pas être utilisés entre eux, par exemple, `1 < 2 > 1` est interdit en PHP. L'expression `1 <= 1 == 1` par contre, est autorisée, car l'opérateur `==` a une précédence inférieure que l'opérateur `<=`.

L'associativité à uniquement du sens pour les opérateurs binaire (et ternaire). Les opérateurs unitaires sont soit préfixé soit suffixé ainsi cette notion n'est pas applicable. Par exemple `!!$a` peut uniquement être groupé de la manière suivante `!(!$a)`.

L'utilisation des parenthèses, y compris lorsqu'elles ne sont pas nécessaires, permet de mieux lire le code en effectuant des groupements explicites plutôt qu'imaginer la priorité des opérateurs et leurs associations.

Le tableau qui suit liste les opérateurs par ordre de priorité, avec la priorité la plus élevée en haut. Les opérateurs sur la même ligne ont une priorité équivalente (donc l'associativité décide du groupement).

Priorité des opérateurs

Associativité	Opérateurs	Information additionnelle
(n/a)	`clone new`	clone et new
droite	`**`	arithmétique
(n/a)	`+ - ++ -- ~ (int) (float) (string) (array) (object) (bool) @`	arithmétique (unitaire + et -), incrément/décrément bit à bit, casting de type et contrôle d'erreur
gauche	`instanceof`	type
(n/a)	`!`	logique
gauche	`* / %`	arithmétique
gauche	`+ - .`	arithmétique (binaire + et -), tableau et chaîne de caractères (. antérieur à PHP 8.0.0)
gauche	`<< >>`	bitwise
gauche	`.`	string (à partir de PHP 8.0.0)
non-associatif	`< <= > >=`	comparaison
non-associatif	`== != === !== <> <=>`	comparaison
gauche	`&`	bitwise et références

Associativité	Opérateurs	Information additionnelle
gauche	`^`	bitwise
gauche	`|`	bitwise
gauche	`&&`	logique
gauche	`||`	logique
droite	`??`	coalescence nul
non-associatif	`? :`	ternaire (gauche--associatif antérieur à PHP 8.0.0)
droite	`= += -= *= **= /= .= %= &= |= ^= <<= >>= ??=`	affectation
(n/a)	`yield from`	yield from
(n/a)	`yield`	yield
(n/a)	`print`	print
gauche	`and`	logique
gauche	`xor`	logique
gauche	`or`	logique

Exemple #1 Associativité

```php
<?php
$a = 3 * 3 % 5; // (3 * 3) % 5 = 4
// L'association des opérateurs ternaires diffère de C/C++
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2 (antérieur à PHP 8.0.0)

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

La priorité et l'association de l'opérateur ne déterminent que la façon dont les expressions sont groupées ; ils ne spécifient pas l'ordre de l'évaluation. PHP ne spécifie pas (de manière générale) l'ordre dans lequel une expression est évaluée et le code qui suppose un ordre spécifique d'évaluation ne devrait pas exister, car le comportement peut changer entre les différentes versions de PHP ou suivant le code environnant.

Exemple #2 Ordre d'évaluation indéfini

```php
<?php
$a = 1;
echo $a + $a++; // peut afficher 2 ou 3

$i = 1;
$array[$i] = $i++; // peut définir l'index 1 ou 2
?>
```

Exemple #3 +, - et . ont la même priorité (antérieur à PHP 8.0.0)

```php
<?php
$x = 4;
// cette ligne peut entraîner une sortie inattendue :
echo "x moins un est égal " . $x-1 . ", en tout cas j'espère\n";
// parce que c'est évalué comme la ligne suivante (antérieur à PHP 8.0.0) :
echo (("x moins un est égal " . $x) - 1) . ", en tout cas j'espère\n";
// la priorité désirée peut être renforcée en utilisant les parenthèses. :
echo "x moins un est égal " . ($x-1) . ", en tout cas j'espère\n";
?>
```

L'exemple ci-dessus va afficher :

```
-1, en tout cas j'espère
-1, en tout cas j'espère
x moins un est égal 3, en tout cas j'espère
```

> **Note**:
>
> Bien que = soit prioritaire sur la plupart des opérateurs, PHP va tout de même exécuter des expressions comme : `if (!$a = foo())`. Dans cette situation, le résultat de `foo()` sera placé dans la variable *$a*.

Historique

Version	Description
8.0.0	La concaténation de chaînes de caractères (`.`) a désormais une précédence moins élevée que l'addition/soustraction arithmétique (`+` et `-`) et les shifts bit-à-bit gauche/droite (`<<` et `>>`); auparavant ceci avait la même précédence que `+` et `-`, et une précédence plus élevée que `<<` et `>>`.
8.0.0	L'opérateur ternaire (`? :`) est désormais non associatif ; auparavant il était gauche-associatif.
7.4.0	Dépendre de la précédence de la concaténation de chaînes de caractères (`.`) relatif à l'addition/soustraction arithmétique (`+` ou `-`) ou les shifts bit-à-bit gauche/droite (`<<` ou `>>`), i.e. les utiliser ensemble dans une expression sans parenthèse, est obsolète.
7.4.0	Dépendre de la gauche-associativité de l'opérateur ternaire (`? :`), c.-à-d. l'imbrication de plusieurs opérateurs ternaires qui ne sont pas entre parenthèse, est obsolète.

User Contributed Notes 9 notes

199
fabmlk ¶
7 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.
```


5
tlili dot mokhtar at gmail dot com ¶
1 year 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
33

aaronw at catalyst dot net dot nz ¶

5 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
51

Carsten Milkau ¶

10 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
8

ivan at dilber dot info ¶

5 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
0

sangala at seznam dot cz ¶

18 days 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
-1
karlisd at gmail dot com ¶
7 years ago

Sometimes it's easier to understand things in your own examples.
If you want to play around operator precedence and look which tests will be made, you can play around with this:

```
<?php
function F($v) {echo $v." "; return false;}
function T($v) {echo $v." "; return true;}

IF (F(0) || T(1) && F(2)  || F(3)  && ! F(4) ) {
  echo "true";
} else echo " false";
?>
```

Now put in IF arguments f for false and t for true, put in them some ID's. Play out by changing "F" to "T" and vice versa, by keeping your ID the same. See output and you will know which arguments actualy were checked.

up
down
-2
instatiendaweb at gmail dot com ¶
1 year 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.
```

up
down
-3
anisgazig at gmail dot com ¶
1 year ago

Three types of operator associativity in php.
1.left
2.rigt
3.non-associativity

Category of three operators are right associativity
1)**
2)=,+=,-=,*=,/=,%=,&=,^=,|=,<<=,>>=,??=,.=
3)??

```
Category of eight operators are non-associativity
1)clone new
2)++,--,~,@
3)!
4)<,<=,>,>=
5)<<,>>
6)yield from
7)yield
8)print

Rest of the operators are left associativity
```

+ add a note