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 »
« Целые числа

- Руководство по PHP
- Справочник языка
- Типы

Change language: Russian

Числа с плавающей точкой

Числа с плавающей точкой или числа с плавающей запятой (известные также как «float», «double» или «real») можно определить следующими синтаксисами:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
$d = 1_234.567; // начиная с PHP 7.4.0
?>
```

Формально начиная с PHP 7.4.0 (ранее подчёркивание не разрешалось):

```
LNUM          [0-9]+(_[0-9]+)*
DNUM          ({LNUM}?"."{LNUM}) | ({LNUM}"."{LNUM}?)
EXPONENT_DNUM (({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

Размер числа с плавающей точкой зависит от платформы, хотя максимум, как правило, составляет 1.8e308 с точностью около 14 десятичных цифр (64-битный формат IEEE).

Внимание

Точность чисел с плавающей точкой

Числа с плавающей точкой имеют ограниченную точность. Хотя это зависит от операционной системы, в PHP обычно используется формат двойной точности IEEE 754, дающий максимальную относительную ошибку округления порядка 1.11e-16. Неэлементарные арифметические операции могут давать большие ошибки, и, разумеется, необходимо принимать во внимание распространение ошибок при совместном использовании нескольких операций.

Кроме того, рациональные числа, которые могут быть точно представлены в виде чисел с плавающей точкой с основанием 10, например, `0.1` или `0.7`, не имеют точного внутреннего представления в качестве чисел с плавающей точкой с основанием 2, вне зависимости от размера мантиссы. Поэтому они и не могут быть преобразованы в их внутреннюю двоичную форму без небольшой потери точности. Это может привести к неожиданным результатам: например, `floor((0.1 + 0.7) * 10)` скорее всего вернёт `7` вместо ожидаемого `8`, так как результат внутреннего представления будет чем-то вроде `7.9999999999999991118...`.

Так что никогда не доверяйте точности чисел с плавающей точкой до последней цифры и не проверяйте напрямую их равенство. Если вам действительно необходима высокая точность, используйте математические функции произвольной точности и gmp-функции.

«Простое» объяснение можно найти в » руководстве по числам с плавающей точкой, которое также называется «Why don't my numbers add up?» («Почему мои числа не складываются?» — англ.)

Преобразование в число с плавающей точкой

Из строк

Если строка содержит число или ведущую числовую последовательность, тогда она будет преобразована в соответствующее значение с плавающей точкой, в противном случае она преобразуется в ноль (`0`).

Из других типов

Для значений других типов преобразование выполняется путём преобразования значения сначала в целое число (int), а затем в число с плавающей точкой (float). Смотрите Преобразование в целое число для получения дополнительной информации.

> **Замечание**:
>
> Поскольку определённые типы имеют неопределённое поведение при преобразовании в целое число (int), то же самое происходит и при преобразовании в число с плавающей точкой (float).

Сравнение чисел с плавающей точкой

Как указано выше, проверять числа с плавающей точкой на равенство проблематично из-за их внутреннего представления. Тем не менее, существуют способы для их сравнения, которые работают несмотря на все эти ограничения.

Для сравнения чисел с плавающей точкой используется верхняя граница относительной ошибки при округлении. Эта величина называется машинной эпсилон или единицей округления (unit roundoff) и представляет собой самую маленькую допустимую разницу при расчётах.

$a и *$b* равны до 5-ти знаков после точки.

```
<?php
$a = 1.23456789;
$b = 1.23456780;
$epsilon = 0.00001;

if (abs($a - $b) < $epsilon) {
echo "true";
}
?>
```

NaN

Некоторые числовые операции могут возвращать значение, представляемое константой **NAN**. Данный результат означает неопределённое или непредставимое значение в операциях с плавающей точкой. Любое строгое или нестрогое сравнение данного значения с другим значением, кроме **true**, включая его самого, возвратит **false**.

Так как **NAN** представляет собой неограниченное количество различных значений, то **NAN** не следует сравнивать с другими значениями, включая её саму. Вместо этого, для определения её наличия необходимо использовать функцию is_nan().

User Contributed Notes 11 notes

251
catalin dot luntraru at gmail dot com ¶
10 years ago

```
$x = 8 - 6.4; // which is equal to 1.6
$y = 1.6;
var_dump($x == $y); // is not true

PHP thinks that 1.6 (coming from a difference) is not equal to 1.6. To make it work, use round()

var_dump(round($x, 2) == round($y, 2)); // this is true

This happens probably because $x is not really 1.6, but 1.599999.. and var_dump shows it to you as being 1.6.
```


111
feline at NOSPAM dot penguin dot servehttp dot com ¶
19 years ago

```
General computing hint: If you're keeping track of money, do yourself and your users the favor of handling everything internally in cents and do as much math as you can in integers. Store values in cents if at all possible. Add and subtract in cents. At every operation that wii involve floats, ask yourself "what will happen in the real world if I get a fraction of a cent here" and if the answer is that this operation will generate a transaction in integer cents, do not try to carry fictional fractional accuracy that will only screw things up later.
```


58
www.sarioz.com ¶
21 years ago

just a comment on something the "Floating point precision" inset, which goes: "This is related to .... 0.3333333."

While the author probably knows what they are talking about, this loss of precision has nothing to do with decimal notation, it has to do with representation as a floating-point binary in a finite register, such as while 0.8 terminates in decimal, it is the repeating 0.110011001100... in binary, which is truncated. 0.1 and 0.7 are also non-terminating in binary, so they are also truncated, and the sum of these truncated numbers does not add up to the truncated binary representation of 0.8 (which is why (floor)(0.8*10) yields a different, more intuitive, result). However, since 2 is a factor of 10, any number that terminates in binary also terminates in decimal.

up
down
1

251701981 at qq dot com ¶

6 months ago

```
<?php

//Please consider the following code
printf("%.53f\n",0.7+0.1); // 0.79999999999999993338661852249060757458209991455078125

var_dump(0.7+0.1); // float(0.8)

var_dump(0.799999999999999); //float(0.8)

var_dump(0.7999999); // float(0.7999999)

//Conclusion: PHP can support up to 53 decimal places, but in some output functions such as var_ Dump, when outputting decimals exceeding 14 places, will round off the 15th place, which causes significant misleading
//experimental environment：linux x64, php7.2.x
```

up
down
19

backov at spotbrokers-nospamplz dot com ¶

20 years ago

I'd like to point out a "feature" of PHP's floating point support that isn't made clear anywhere here, and was driving me insane.

This test (where var_dump says that $a=0.1 and $b=0.1)

```
if ($a>=$b) echo "blah!";
```

Will fail in some cases due to hidden precision (standard C problem, that PHP docs make no mention of, so I assumed they had gotten rid of it). I should point out that I originally thought this was an issue with the floats being stored as strings, so I forced them to be floats and they still didn't get evaluated properly (probably 2 different problems there).

To fix, I had to do this horrible kludge (the equivelant of anyway):

```
if (round($a,3)>=round($b,3)) echo "blah!";
```

THIS works. Obviously even though var_dump says the variables are identical, and they SHOULD BE identical (started at 0.01 and added 0.001 repeatedly), they're not. There's some hidden precision there that was making me tear my hair out. Perhaps this should be added to the documentation?

up
down
5

lwiwala at gmail dot com ¶

6 years ago

To compare two numbers use:

```
$epsilon = 1e-6;

if(abs($firstNumber-$secondNumber) < $epsilon){
// equals
}
```

up
down
8
Luzian ¶
18 years ago
Be careful when using float values in strings that are used as code later, for example when generating JavaScript code or SQL statements. The float is actually formatted according to the browser's locale setting, which means that "0.23" will result in "0,23". Imagine something like this:

```
$x = 0.23;
$js = "var foo = doBar($x);";
print $js;
```

This would result in a different result for users with some locales. On most systems, this would print:

```
var foo = doBar(0.23);
```

but when for example a user from Germany arrives, it would be different:

```
var foo = doBar(0,23);
```

which is obviously a different call to the function. JavaScript won't state an error, additional arguments are discarded without notice, but the function doBar(a) would get 0 as parameter. Similar problems could arise anywhere else (SQL, any string used as code somewhere else). The problem persists, if you use the "." operator instead of evaluating the variable in the string.

So if you REALLY need to be sure to have the string correctly formatted, use number_format() to do it!
up
down
6
james dot cridland at virginradio dot co dot uk ¶
20 years ago
The 'floating point precision' box in practice means:

```
<? echo (69.1-floor(69.1)); ?>
Think this'll return 0.1?
It doesn't - it returns 0.099999999999994

<? echo round((69.1-floor(69.1))); ?>
This returns 0.1 and is the workaround we use.
```

Note that
```
<? echo (4.1-floor(4.1)); ?>
```
does return 0.1 - so if you, like us, test this with low numbers, you won't, like us, understand why all of a sudden your script stops working, until you spend a lot of time, like us, debugging it.

So, that's all lovely then.
up
down
6
magicaltux at php dot net ¶
13 years ago
In some cases you may want to get the maximum value for a float without getting "INF".

var_dump(1.8e308); will usually show: float(INF)

I wrote a tiny function that will iterate in order to find the biggest non-infinite float value. It comes with a configurable multiplicator and affine values so you can share more CPU to get a more accurate estimate.

I haven't seen better values with more affine, but well, the possibility is here so if you really thing it's worth the cpu time, just try to affine more.

Best results seems to be with mul=2/affine=1. You can play with the values and see what you get. The good thing is this method will work on any system.

```
<?php
function float_max($mul = 2, $affine = 1) {
$max = 1; $omax = 0;
while((string)$max != 'INF') { $omax = $max; $max *= $mul; }

for($i = 0; $i < $affine; $i++) {
$pmax = 1; $max = $omax;
while((string)$max != 'INF') {
$omax = $max;
$max += $pmax;
$pmax *= $mul;
}
}
return $omax;
}
?>
```

up
down
3

zelko at mojeime dot com ¶
12 years ago

```
<?php
$binarydata32 = pack('H*','00000000');
$float32 = unpack("f", $binarydata32); // 0.0

$binarydata64 = pack('H*','0000000000000000');
$float64 = unpack("d", $binarydata64); // 0.0
?>
```

I get 0 both for 32-bit and 64-bit numbers.

But, please don't use your own "functions" to "convert" from float to binary and vice versa. Looping performance in PHP is horrible. Using pack/unpack you use processor's encoding, which is always correct. In C++ you can access the same 32/64 data as either float/double or 32/64 bit integer. No "conversions".

To get binary encoding:

```
<?php
$float32 = pack("f", 5300231);
$binarydata32 =unpack('H*',$float32); //"0EC0A14A"

$float64 = pack("d", 5300231);
$binarydata64 =unpack('H*',$float64); //"000000C001385441"
?>
```

And my example from half a year ago:

```
<?php
$binarydata32 = pack('H*','0EC0A14A');
$float32 = unpack("f", $binarydata32); // 5300231

$binarydata64 = pack('H*','000000C001385441');
$float64 = unpack("d", $binarydata64); // 5300231
?>
```

And please mind the Big and Little endian boys...

up
down
3

rick at ninjafoo dot com ¶
18 years ago

```
Concider the following:

(19.6*100) != 1960

echo gettype(19.6*100) returns 'double', However even .....

(19.6*100) !== (double)1960

19.6*100 cannot be compaired to anything without manually
casting it as something else first.

(string)(19.6*100) == 1960

Rule of thumb, if it has a decimal point, use the BCMath functions.
```