Integers ¶

An int is a number of the set ℤ = {..., -2, -1, 0, 1, 2, ...}.

See also:

- Arbitrary length integer / GMP
- Floating point numbers
- Arbitrary precision / BCMath

Syntax ¶

ints can be specified in decimal (base 10), hexadecimal (base 16), octal (base 8) or binary (base 2) notation. The negation operator can be used to denote a negative int.

To use octal notation, precede the number with a `0` (zero). To use hexadecimal notation precede the number with `0x`. To use binary notation precede the number with `0b`.

As of PHP 7.4.0, integer literals may contain underscores (_) between digits, for better readability of literals. These underscores are removed by PHP's scanner.

Example #1 Integer literals

```
<?php
$a = 1234; // decimal number
$a = 0123; // octal number (equivalent to 83 decimal)
$a = 0x1A; // hexadecimal number (equivalent to 26 decimal)
$a = 0b11111111; // binary number (equivalent to 255 decimal)
$a = 1_234_567; // decimal number (as of PHP 7.4.0)
?>
```

Formally, the structure for int literals is as of PHP 7.4.0 (previously, underscores have not been allowed):

```
decimal     : [1-9][0-9]*(_[0-9]+)*
            | 0

hexadecimal : 0[xX][0-9a-fA-F]+(_[0-9a-fA-F]+)*

octal       : 0[0-7]+(_[0-7]+)*

binary      : 0[bB][01]+(_[01]+)*

integer     : decimal
            | hexadecimal
            | octal
            | binary
```

The size of an int is platform-dependent, although a maximum value of about two billion is the usual value (that's 32 bits signed). 64-bit platforms usually have a maximum value of about 9E18. PHP does not support unsigned ints. int size can be determined using the constant **`PHP_INT_SIZE`**, maximum value using the constant **`PHP_INT_MAX`**, and minimum value using the constant **`PHP_INT_MIN`**.

Integer overflow ¶

If PHP encounters a number beyond the bounds of the int type, it will be interpreted as a float instead. Also, an operation which results in a number beyond the bounds of the int type will return a float instead.

Example #2 Integer overflow on a 32-bit system

```php
<?php
$large_number = 2147483647;
var_dump($large_number);                     // int(2147483647)

$large_number = 2147483648;
var_dump($large_number);                     // float(2147483648)

$million = 1000000;
$large_number =  50000 * $million;
var_dump($large_number);                     // float(50000000000)
?>
```

Example #3 Integer overflow on a 64-bit system

```php
<?php
$large_number = 9223372036854775807;
var_dump($large_number);                     // int(9223372036854775807)

$large_number = 9223372036854775808;
var_dump($large_number);                     // float(9.2233720368548E+18)

$million = 1000000;
$large_number =  50000000000000 * $million;
var_dump($large_number);                     // float(5.0E+19)
?>
```

There is no int division operator in PHP, to achieve this use the intdiv() function. `1/2` yields the float `0.5`. The value can be cast to an int to round it towards zero, or the round() function provides finer control over rounding.

```php
<?php
var_dump(25/7);         // float(3.5714285714286)
var_dump((int) (25/7)); // int(3)
var_dump(round(25/7));  // float(4)
?>
```

Converting to integer ¶

To explicitly convert a value to int, use either the `(int)` or `(integer)` casts. However, in most cases the cast is not needed, since a value will be automatically converted if an operator, function or control structure requires an int argument. A value can also be converted to int with the intval() function.

If a resource is converted to an int, then the result will be the unique resource number assigned to theresource by PHP at runtime.

See also Type Juggling.

From booleans ¶

`false` will yield `0` (zero), and **`true`** will yield `1` (one).

From floating point numbers ¶

When converting from float to int, the number will be rounded *towards zero*.

If the float is beyond the boundaries of int (usually `+/- 2.15e+9 = 2^31` on 32-bit platforms and `+/- 9.22e+18 = 2^63` on 64-bit platforms), the result is undefined, since the float doesn't have enough precision to give an exact int result. No warning, not even a notice will be issued when this happens!

> **Note**:
>
> NaN and Infinity will always be zero when cast to int.

Warning

Never cast an unknown fraction to int, as this can sometimes lead to unexpected results.

```
<?php
echo (int) ( (0.1+0.7) * 10 ); // echoes 7!
?>
```

See also the warning about float precision.

From strings ¶

If the string is numeric or leading numeric then it will resolve to the corresponding integer value, otherwise it is converted to zero (`0`).

From NULL ¶

`null` is always converted to zero (`0`).

From other types ¶

Caution

The behaviour of converting to int is undefined for other types. Do *not* rely on any observed behaviour, as it can change without notice.