PHP - Integers

Integer is one of the built-in scalar types in PHP. A whole number, without a decimal point in the literal, is of the type "int" in PHP. An integer can be represented in decimal (base 10), hexadecimal (base 16), octal (base 8) or binary (base 2) notation.

To use octal notation, a number is preceded with "0o" or "0O" (PHP 8.1.0 and earlier). From PHP 8.1.0 onwards, a number prefixed with "0" and without a decimal point is an octal number.

To use hexadecimal notation, precede the number with "0x". To use binary notation, precede the number with "0b".

Example

```php
<?php
   $a = 1234;
   echo "1234 is an Integer in decimal notation: $a\n";

   $b = 0123;
   echo "0o123 is an integer in Octal notation: $b\n";

   $c = 0x1A;
   echo "0xaA is an integer in Hexadecimal notation: $c\n";

   $d = 0b1111;
   echo "0b1111 is an integer in binary notation: $d\n";
?>
```

It will produce the following **output** –

```
1234 is an Integer in decimal notation: 1234
0o123 is an integer in Octal notation: 83
0xaA is an integer in Hexadecimal notation: 26
0b1111 is an integer in binary notation: 15
```

PHP 7.4.0 onwards, integer literals may contain underscores (_) as separators between digits, for better readability of literals. These underscores are removed by

PHP's scanner.

Example

```
<?php
   $a = 1_234_567;
   echo "1_234_567 is an Integer with _ as separator: $a\n";
?>
```

It will produce the following **output** –

```
1_234_567 is an Integer with _ as separator: 1234567
```

PHP does not support unsigned ints. The size of an **int** is platform dependent. On 32 bit systems, the maximum value is about two billion. 64-bit platforms usually have a maximum value of about 9E18.

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.

If an integer number happens to be beyond the bounds of the **int** type, or any operation results in a number beyond the bounds of the **int** type, it will be interpreted as a float instead.

Example

```
<?php
   $x = 1000000;
   $y =  50000000000000 * $x;
   var_dump($y);
?>
```

It will produce the following **output** –

```
float(5.0E+19)
```

PHP doesn't have any operator for integer division. Hence, a division operation between an integer and a float always results in float. To obtain integral division, you may use the **intval()** built-in function.

Example

```
<?php
   $x = 10;
   $y = 3.5;
   $z = $x/$y;
   var_dump ($z);
   $z = intdiv($x, $y);
   var_dump ($z);
?>
```

It will produce the following **output** –

```
float(2.857142857142857)
int(3)
```