PHP – Maths Functions

To enable mathematical operations, PHP has mathematical (arithmetic) operators and a number of mathematical functions. In this chapter, the following mathematical functions are explained with examples.

PHP abs() Function

The abs() function is an in-built function in PHP iterpreter. This function accepts any number as argument and returns a positive value, disregarding its sign. Absolute value of any number is always positive.

```
abs( mixed $num)
```

PHP abs() function returns the absolute value of **num**. If the data type of num is float, its return type will also be float. For integer parameter, the return type is integer.

Example

Take a look at this following example –

Open Compiler

```
<?php
   $num=-9.99;
   echo "negative float number: " . $num . "\n";
   echo "absolute value : " . abs($num) . "\n";

   $num=25.55;
   echo "positive float number: " . $num . "\n";
   echo "absolute value : " . abs($num). "\n";

   $num=-45;
   echo "negative integer number: " . $num . "\n";
   echo "absolute value : " . abs($num) . "\n";

   $num=25;
   echo "positive integer number: " . $num . "\n";
```

```
    echo "absolute value : " . abs($num);
?>
```

It will produce the following **output** –

```
negative float number: -9.99
absolute value : 9.99
positive float number: 25.55
absolute value : 25.55
negative integer number: -45
absolute value : 45
positive integer number: 25
absolute value : 25
```

PHP ceil() Function

The ceil() function is an in-built function in PHP iterpreter. This function accepts any float number as argument and rounds it up to the next highest integer. This function always returns a float number as the range of float is bigger than that of integer.

```
ceil ( float $num ) : float
```

PHP ceil() function returns the smallest integer value that is bigger than or equal to given parameter.

Example 1

The following code rounds 5.78 to its next highest integer which is 6

Open Compiler

```
<?php
   $arg=5.78;
   $val=ceil($arg);
   echo "ceil(" . $arg .  ") = " . $val;
?>
```

It will produce the following **output** –

```
ceil(5.78) = 6
```

Example 2

The following example shows how you can find the next highest integer of 15.05.

Open Compiler

```php
<?php
   $arg=15.05;
   $val=ceil($arg);
   echo "ceil(" . $arg .  ") = " . $val;
?>
```

It will produce the following **output** –

```
ceil(15.05) = 16
```

Example 3

For negative number, it is rounded towards 0.

Open Compiler

```php
<?php
   $arg=-3.95;
   $val=ceil($arg);
   echo "ceil(" . $arg .  ") = " . $val;
?>
```

It will produce the following **output** –

```
ceil(-3.95) = -3
```

PHP exp() Function

The **exp()** function calculates exponent of e that is Euler Number. PHP has a predefined constant M_E that represents Euler Number and is equal to 2.7182818284590452354. Hence, exp(x) returns 2.7182818284590452354x

This function always returns a float.

```
exp ( float $arg ) : float
```

PHP exp() function returns the Euler Number e raised to given **arg.** Note that **e** is the base of natural algorithm. The exp() function is the inverse of natural logarithm.

Example 1

One of the predefined constants in PHP is **M_LN2** which stands for loge2 and is equal to 0.69314718055994530942. So, the exp() of this value will return 2.

Open Compiler

```php
<?php
   echo "exp(" . M_LN2 . ") = " . exp(M_LN2);
?>
```

It will produce the following **output** –

```
exp(0.69314718055995) = 2
```

Example 2

M_LN10 is another predefined constant representing loge10. This program calculates exp(M_LN10) and returns 10.

Open Compiler

```php
<?php
   echo "exp(" . M_LN10 . ") = " . exp(M_LN10);
?>
```

It will produce the following **output** –

```
exp(2.302585092994) = 10
```

PHP floor() Function

The floor() function is another in-built function in PHP interpreter. This function accepts any float number as argument and rounds it down to the next lowest integer. This function always returns a float number as the range of float is bigger than that of integer.

```
floor ( float $num ) : float
```

PHP floor() function returns the largest integer less than or equal to the given parameter.

Example 1

The following example shows how to round 15.05 to its next highest integer which is 15

Open Compiler

```
<?php
   $arg=15.05;
   $val=floor($arg);
   echo "floor(" . $arg .  ") = " . $val;
?>
```

It will produce the following **output** –

```
floor(15.05) = 15
```

Example 2

The following example shows how to find the next lowest integer of 5.78.

Open Compiler

```
<?php
   $arg=5.78;
   $val=floor($arg);
   echo "floor(" . $arg .  ") = " . $val;
?>
```

It will produce the following **output** –

```
floor(5.78) = 5
```

Example 3

Negative numbers are rounded away from 0.

</> Open Compiler

```
<?php
   $arg=-3.95;
   $val=floor($arg);
   echo "floor(" . $arg .  ") = " . $val;
?>
```

It will produce the following **output** –

```
floor(-3.95) = -4
```

PHP intdiv() Function

The intdiv() function returns the integer quotient of two integer parameters. If x/y results in "i" as division and "r" as remainder, then –

```
x = y*i+r
```

In this case, intdiv(x,y) returns "i"

```
intdiv ( int $x , int $y ) : int
```

The "x" parameter forms numerator part of the division expression, while the "y" parameter forms the denominator part of the division expression.

PHP intdiv() function returns the integer quotient of division of "x" by "y". The return value is positive if both the parameters are positive or both the parameters are negative.

Example 1

The following example shows that if the numerator is less than the denominator, then intdiv() function returns 0.

Open Compiler

```
<?php
   $x=10;
   $y=3;
   $r=intdiv($x, $y);
   echo "intdiv(" . $x . "," . $y . ") = " . $r . "\n";
   $r=intdiv($y, $x);
   echo "intdiv(" . $y . "," . $x . ") = " . $r;
?>
```

It will produce the following **output** −

```
intdiv(10,3) = 3
intdiv(3,10) = 0
```

Example 2

In the following example, intdiv() function returns a negative integer because either the numerator or the denominator is negative.

Open Compiler

```
<?php
   $x=10;
   $y=3;
   $r=intdiv($x, $y);
   echo "intdiv(" . $x . "," . $y . ") = " . $r . "\n";

   $x=10;
   $y=-3;
   $r=intdiv($x, $y);
   echo "intdiv(" . $x . "," . $y . ") = " . $r . "\n";

   $x=-10;
   $y=3;
   $r=intdiv($x, $y);
   echo "intdiv(" . $x . "," . $y . ") = " . $r . "\n";

   $x=-10;
   $y=-3;
   $r=intdiv($x, $y);
   echo "intdiv(" . $x . "," . $y . ") = " . $r ;
?>
```

It will produce the following **output** −

```
intdiv(10,3) = 3
intdiv(10,-3) = -3
intdiv(-10,3) = -3
intdiv(-10,-3) = 3
```

Example 3

Denominator is 0 in the following example. It results in DivisionByZeroError exception.

Open Compiler

```
<?php
   $x=10;
   $y=0;
   $r=intdiv($x, $y);
   echo "intdiv(" . $x . "," . $y . ") = " . $r . "\n";
?>
```

It will produce the following **output** −

```
PHP Fatal error:  Uncaught DivisionByZeroError: Division by zero
```

Example 4

The fractional parts in both the parameters are ignored. PHP intdiv() function is applied only to the integer parts.

Open Compiler

```
<?php
   $x=2.90;
   $y=1.90;
   $r=intdiv($x, $y);
```

```
    echo "intdiv(" . $x . "," . $y . ") = " . $r . "";
?>
```

It will produce the following **output** −

```
intdiv(2.9,1.9) = 2
```

PHP log10() Function

The **log10** () function calculates the base-10 logarithm of a number. Base-10 logarithm is also called **common** or **standard algorithm**. The log10(x) function calculates log10x. It is related to natural algorithm by the following equation −

```
log10x=logex/loge10 ; So that
log10100=loge100/loge10 = 2
```

In PHP, **log10** is represented by **log10()** function

```
log10 ( float $arg ) : float
```

PHP log10() function returns the base-10 logarithm of **arg**.

Example 1

The following code calculates the base-10 logarithm of 100

```
<?php
    $arg=100;
    echo "log10(" . $arg. ")=" . log10($arg) . "";
?>
```

It will produce the following **output** −

```
log10(100)=2
```

Example 2

The following code calculates the base-10 logarithm of Euler Number **M_E**. The result is equal to a predefined constant **M_LOG10E**

</> Open Compiler

```
<?php
   $arg=M_E;
   echo "log10(" . $arg. ")=" . log10($arg) . "\n";
   echo "predefined constant M_LOG10E=" . M_LOG10E;
?>
```

It will produce the following **output** −

```
log10(2.718281828459)=0.43429448190325
predefined constant M_LOG10E=0.43429448190325
```

Example 3

The following code calculates log100 and returns -∞.

</> Open Compiler

```
<?php
   $arg=0;
   echo "log10(" . $arg. ")=" . log10($arg) . "";
?>
```

It will produce the following **output** −

```
log10(0)=-INF
```

Example 4

Similarly sqrt(-1) results in NAN. Hence, its log10() also returns NAN.

</> Open Compiler

```
<?php
   $arg=sqrt(-1);
```

```
    echo "log10(" . $arg. ")=" . log10($arg) . "";
?>
```

It will produce the following **output** –

```
log10(NAN)=NAN
```

PHP max() Function

The max () function returns the highest element in an array, or the highest amongst two or more comma separated parameters.

```
max ( array $values ) : mixed
```

Or,

```
max ( mixed $value1 [, mixed $... ] ) : mixed
```

- If only one parameter is given, it should be an array of values which may be of same or different types.
- If two or more parameters are given, they should be any comparable values of same or different types.

PHP max() function returns the highest value from the array parameter or sequence of values. Standard comparison operators are applicable. If multiple values of different types evaluate as equal (e.g. 0 and 'PHP'), the first parameter to the function will be returned.

Example 1

The following code returns the highest value from a numeric array.

</> Open Compiler

```
<?php
    $arg=array(23, 5.55, 142, 56, 99);
    echo "array=";
    foreach ($arg as $i) echo $i . ",";
    echo "\n";
    echo "max = " . max($arg);
?>
```

It will produce the following **output** –

```
array=23,5.55,142,56,99,
max = 142
```

Example 2

The following code returns max() from an array of strings.

</> Open Compiler

```php
<?php
   $arg=array("Java", "Angular", "PHP", "C", "Kotlin");
   echo "array=";
   foreach ($arg as $i) echo $i . ",";
   echo "\n";
   echo "max = " . max($arg);
?>
```

It will produce the following **output** –

```
array=Java,Angular,PHP,C,Kotlin,
max = PHP
```

Example 3

In the following example, a series of string values is provided to the max() function. Let's see how it behaves –

</> Open Compiler

```php
<?php
   $val1="Java";
   $val2="Angular";
   $val3="PHP";
   $val4="C";
   $val5="Kotlin";
   echo "values=" . $val1 . "," . $val2 . "," . $val3 . "," .   $val4 . "," . $val
   echo "max = " . max($val1, $val2, $val3,$val4,$val5);
?>
```

It will produce the following **output** –

```
values=Java,Angular,PHP,C,Kotlin
max = PHP
```

Example 4

In this example, the given array is a collection of mixed data types.

Open Compiler

```
<?php
   $arg=array(23, "Java", 142, 1e2, 99);
   echo "array=";
   foreach ($arg as $i) echo $i . ",";
   echo "\n";
   echo "max = " . max($arg);
?>
```

It will produce the following **output** –

```
array=23,Java,142,100,99,
max = 142
```

PHP min() Function

The min () function returns the lowest element in an array, or the lowest amongst two or more comma separated parameters.

```
min ( array $values ) : mixed
```

Or,

```
min ( mixed $value1 [, mixed $... ] ) : mixed
```

- If only one parameter is given, it should be an array of values which may be of same or different types
- If two or more parameters are given, they should be any comparable values of same or different types

PHP min() function returns the lowest value from the array parameter or sequence of values. Standard comparison operators are applicable. If multiple values of different types evaluate as equal (e.g. 0 and 'PHP'), the first parameter to the function will be returned

Example 1

The following code returns the smallest value from numeric array.

</> Open Compiler

```
<?php
   $arg=array(23, 5.55, 142, 56, 99);
   echo "array=";
   foreach ($arg as $i) echo $i . ",";
   echo "\n";
   echo "min = " . min($arg);
?>
```

It will produce the following **output** –

```
array=23,5.55,142,56,99,
min = 5.55
```

Example 2

The following code returns min() from an array of strings.

</> Open Compiler

```
<?php
   $arg=array("Java", "Angular", "PHP", "C", "Kotlin");
   echo "array=";
   foreach ($arg as $i) echo $i . ",";
   echo "\n";
   echo "min = " . min($arg);
?>
```

It will produce the following **output** –

```
array=Java,Angular,PHP,C,Kotlin,
min = Angular
```

Example 3

In this example, a series of string values is provided to the min() function.

Open Compiler

```
<?php
   $val1="Java";
   $val2="Angular";
   $val3="PHP";
   $val4="C";
   $val5="Kotlin";
   echo "values=" . $val1 . "," . $val2 . "," . $val3 . "," .   $val4 . "," . $val
   echo "min = " . min($val1, $val2, $val3,$val4,$val5);
?>
```

It will produce the following **output** –

```
values=Java,Angular,PHP,C,Kotlin
min = Angular
```

Example 4

In this example, the given array is a collection of mixed data types.

Open Compiler

```
<?php
   $arg=array(23, "Java", 142, 1e2, 99);
   echo "array=";
   foreach ($arg as $i) echo $i . ",";
```

```
    echo "\n";
    echo "min = " . min($arg);
?>
```

It will produce the following **output** –

```
array=23,Java,142,100,99,
min = 23
```

PHP pow() Function

The **pow** () function is used to compute the power of a certain number. It returns xy calculation, also termed as x raised to y. PHP also provides "**" as exponentiation operator.

So, pow(x,y) returns xy which is same as x**y.

```
pow ( number $base , number $exp ) : number
```

The first parameter is the base to be raised. The second parameter is the power to which base needs to be raised.

PHP pow() function returns the base raised to the power of exp. If both arguments are non-negative integers, the result is returned as integer, otherwise it is returned as a float.

Example 1

The following example calculates 102 using pow() function –

Open Compiler

```
<?php
    echo "pow(10,2) = " . pow(10,2);
    echo " using ** operator " . 10**2;
?>
```

It will produce the following **output** –

```
pow(10,2) = 100 using ** operator 100
```

Example 2

Any number raised to 0 results in 1. This is verified in the following example –

```php
<?php
   $x=10;
   $y=0;
   echo "pow(" . $x, "," . $y . ")=". pow($x,$y);
?>
```

It will produce the following **output** –

```
pow(10,0)=1
```

Example 3

The following example shows how you can compute the square root of 100 using the pow() function –

```php
<?php
   $x=100;
   $y=0.5;
   echo "pow(" . $x, "," . $y . ")=". pow($x,$y) . "\n";
   echo "using sqrt() function : ". sqrt(100);
?>
```

It will produce the following **output** –

```
pow(100,0.5)=10
using sqrt() function : 10
```

Example 4

This example shows how you can use the pow() function to calculate the area of a circle.

</> Open Compiler

```
<?php
   $radius=5;
   echo "radius = " . $radius . " area = " . M_PI*pow(5,2);
?>
```

It will produce the following **output** –

```
radius = 5 area = 78.539816339745
```

PHP round() Function

The round() function proves useful in rounding any floating point number upto a desired precision level. Positive precision parameter causes the number to be rounded after the decimal point; whereas with negative precision, rounding occurs before the decimal point. Precision is "0" by default.

For example, round(10.6) returns 11, round(10.2) returns 10. The function always returns a floating point number.

This function also has another optional parameter called **mode** that takes one of the redefined constants described later.

```
round ( float $value , int $precision , int $mode ) : float
```

Parameters

- **Value** – A float number to be rounded.
- **Precision** – Number of decimal digits to round to. Default is 0. Positive precision rounds given number after decimal point. Negative precision rounds the given number before decimal point.
- **Mode** – One of the following predefined constants.

Sr.No	Constant & Description
1	**PHP_ROUND_HALF_UP** Rounds number away from 0 when it is half way there. Hence, 1.5 becomes 2 and -1.5 to -2
2	**PHP_ROUND_HALF_DOWN**

	Rounds number towards 0 when it is half way there. Hence 1.5 becomes 1 and -1.5 to -1
3	**PHP_ROUND_HALF_EVEN** Rounds the number to nearest even value
4	**PHP_ROUND_HALF_ODD** Rounds the number to nearest odd value

PHP round() function returns a float number that by rounding the value to a desired precision.

Example 1

The following code rounds the given number to positive precision values −

```
<?php
   $arg=1234.567;
   echo "round(" . $arg . ") = " . round($arg) . "\n";
   echo "round(" . $arg . ",1) = " . round($arg,1) . "\n";
   echo "round(" . $arg . ",2) = " . round($arg,2) . "";
?>
```

It will produce the following **output** −

```
round(1234.567) = 1235
round(1234.567,1) = 1234.6
round(1234.567,2) = 1234.57
```

Example 2

The following code rounds the number to negative precision values −

```
<?php
   $arg=1234.567;
   echo "round(" . $arg . ") = " . round($arg) . "\n";
   echo "round(" . $arg . ",-1) = " . round($arg,-1) . "\n";
   echo "round(" . $arg . ",-2) = " . round($arg,-2) . "";
?>
```

It will produce the following **output** –

```
round(1234.567) = 1235
round(1234.567,-1) = 1230
round(1234.567,-2) = 1200
```

Example 3

The following code uses UP and DOWN mode constants for rounding –

Open Compiler

```
<?php
   echo "round(3.45,HALF_UP) = " . round(3.45,0, PHP_ROUND_HALF_UP) . "\n";
   echo "round(3.75 HALF_UP) = " . round(3.75, 1, PHP_ROUND_HALF_DOWN) . "";
?>
```

It will produce the following **output** –

```
round(3.45,HALF_UP) = 3
round(3.75 HALF_UP) = 3.7
```

Example 4

The following code uses ODD and EVEN modes for rounding –

Open Compiler

```
<?php
   echo "round( 3.45,HALF_ODD) = " . round(3.45,0, PHP_ROUND_HALF_ODD) . "\n";
   echo "round(3.78 HALF_EVEN) = " . round(3.78, 0, PHP_ROUND_HALF_EVEN) . "";
?>
```

It will produce the following **output** –

```
round(3.45,HALF_ODD) = 3
round(3.78, HALF_EVEN) = 4
```

PHP sqrt() Function

The sqrt() function returns the square root of a positive float number. Since square root for a negative number is not defined, it returns NAN. This is one of the most commonly used functions. This function always returns a floating point number.

```
sqrt (float $arg) : float
```

PHP sqrt() function returns the square root of the given arg number. For negative numbers, the function returns NAN.

Example 1

The following code calculates the square root of 100 −

Open Compiler

```php
<?php
   $arg = 100;
   echo "Square root of " . $arg . "=" . sqrt($arg) . "";
?>
```

It will produce the following **output** −

```
Square root of 100=10
```

Example 2

For sqrt(2), 1/sqrt(2) and sqrt(3), PHP has special predefined constants M_SQRT2, M_SQRT1_2 and M_SQRT3, respectively.

Open Compiler

```php
<?php
   echo "sqrt(2) = " . sqrt(2) . "\n";
   echo "M_SQRT2 = " . M_SQRT2. "\n";
   echo "sqrt(3) = " . sqrt(3) . "\n";
   echo "M_SQRT3 = " . M_SQRT3 . "\n";
   echo "1/sqrt(2)) = " . 1/sqrt(2) . "\n";
   echo "M_SQRT1_2 = " . M_SQRT1_2 . "";
?>
```

It will produce the following **output** –

```
sqrt(2) = 1.4142135623731
M_SQRT2 = 1.4142135623731
sqrt(3) = 1.7320508075689
M_SQRT3 = 1.7320508075689
1/sqrt(2)) = 0.70710678118655
M_SQRT1_2 = 0.70710678118655
```

Example 3

The mathematical constants M_SQRTPI and M_2_SQRTPI represent values of sqrt(Π) and 2/sqrt(Π).

</> Open Compiler

```
<?php
   echo "sqrt(pi) = " . sqrt(M_PI) . "\n";
   echo "M_SQRTPI = " . M_SQRTPI. "\n";
   echo "2/sqrt(pi) = " . 2/sqrt(M_PI) . "\n";
   echo "M_2_SQRTPI = " . M_2_SQRTPI . "";
?>
```

It will produce the following **output** –

```
sqrt(pi) = 1.7724538509055
M_SQRTPI = 1.7724538509055
2/sqrt(pi) = 1.1283791670955
M_2_SQRTPI = 1.1283791670955
```

Example 4

sqrt(-1) is undefined, hence it returns NAN.

</> Open Compiler

```
<?php
   echo "sqrt(-1) = " . sqrt(-1) . "";
?>
```

It will produce the following **output** −

```
sqrt(-1) = NAN
```

Predefined Mathematical Constants

In addition to the above mathematical functions, PHP also has the following list of predefined mathematical constants −

Constant	Value	Description
M_PI	3.14159265358979323846	Pi
M_E	2.7182818284590452354	Euler Number e
M_LOG2E	1.4426950408889634074	log2 e
M_LOG10E	0.43429448190325182765	log10 e
M_LN2	0.69314718055994530942	loge 2
M_LN10	M_LN10 2.30258509299404568402 loge 10	loge 10
M_PI_2	1.57079632679489661923	pi/2
M_PI_4	0.78539816339744830962	pi/4
M_1_PI	0.31830988618379067154	1/pi
M_2_PI	0.63661977236758134308	2/pi
M_SQRTPI	1.77245385090551602729	sqrt(pi)
M_2_SQRTPI	1.12837916709551257390	2/sqrt(pi)
M_SQRT2	1.41421356237309504880	sqrt(2)
M_SQRT3	1.73205080756887729352	sqrt(3)
M_SQRT1_2	0.70710678118654752440	1/sqrt(2)

M_LNPI	1.1447298858494001741 4	loge(pi)
M_EULER	0.57721566490153286061	Euler constant
PHP_ROUND_HALF_UP	1	Round halves up
PHP_ROUND_HALF_DOWN	2	Round halves down
PHP_ROUND_HALF_EVEN	3	Round halves to even numbers
PHP_ROUND_HALF_ODD	4	Round halves to odd numbers
NAN	NAN	Not A Number
INF	INF	Infinity