## Operator Precedence [¶](http://php.net/manual/en/language.operators.precedence.php#language.operators.precedence)

The precedence of an operator specifies how "tightly" it binds two expressions together. For example, in the expression 1 + 5 \* 3, the answer is 16 and not 18 because the multiplication ("\*") operator has a higher precedence than the addition ("+") operator. Parentheses may be used to force precedence, if necessary. For instance: (1 + 5) \* 3 evaluates to 18.

When operators have equal precedence their associativity decides how the operators are grouped. For example "-" is left-associative, so 1 - 2 - 3 is grouped as (1 - 2) - 3 and evaluates to -4. "=" on the other hand is right-associative, so $a = $b = $c is grouped as $a = ($b = $c).

Operators of equal precedence that are non-associative cannot be used next to each other, for example 1 < 2 > 1 is illegal in PHP. The expression 1 <= 1 == 1 on the other hand is legal, because the == operator has lesser precedence than the <= operator.

Use of parentheses, even when not strictly necessary, can often increase readability of the code by making grouping explicit rather than relying on the implicit operator precedence and associativity.

The following table lists the operators in order of precedence, with the highest-precedence ones at the top. Operators on the same line have equal precedence, in which case associativity decides grouping.

| **Operator Precedence** | | |
| --- | --- | --- |
| **Associativity** | **Operators** | **Additional Information** |
| non-associative | clone new | [clone](http://php.net/manual/en/language.oop5.cloning.php) and [new](http://php.net/manual/en/language.oop5.basic.php#language.oop5.basic.new) |
| left | [ | [array()](http://php.net/manual/en/function.array.php) |
| left | \*\* | [arithmetic](http://php.net/manual/en/language.operators.arithmetic.php) |
| right | ++ -- ~ (int) (float) (string) (array) (object) (bool) @ | [types](http://php.net/manual/en/language.types.php) and [increment/decrement](http://php.net/manual/en/language.operators.increment.php) |
| non-associative | instanceof | [types](http://php.net/manual/en/language.types.php) |
| right | ! | [logical](http://php.net/manual/en/language.operators.logical.php) |
| left | \* / % | [arithmetic](http://php.net/manual/en/language.operators.arithmetic.php) |
| left | + - . | [arithmetic](http://php.net/manual/en/language.operators.arithmetic.php) and [string](http://php.net/manual/en/language.operators.string.php) |
| left | << >> | [bitwise](http://php.net/manual/en/language.operators.bitwise.php) |
| non-associative | < <= > >= | [comparison](http://php.net/manual/en/language.operators.comparison.php) |
| non-associative | == != === !== <> | [comparison](http://php.net/manual/en/language.operators.comparison.php) |
| left | & | [bitwise](http://php.net/manual/en/language.operators.bitwise.php) and [references](http://php.net/manual/en/language.references.php) |
| left | ^ | [bitwise](http://php.net/manual/en/language.operators.bitwise.php) |
| left | | | [bitwise](http://php.net/manual/en/language.operators.bitwise.php) |
| left | && | [logical](http://php.net/manual/en/language.operators.logical.php) |
| left | || | [logical](http://php.net/manual/en/language.operators.logical.php) |
| left | ? : | [ternary](http://php.net/manual/en/language.operators.comparison.php#language.operators.comparison.ternary) |
| right | = += -= \*= /= .= %= &= |= ^= <<= >>= => | [assignment](http://php.net/manual/en/language.operators.assignment.php) |
| left | and | [logical](http://php.net/manual/en/language.operators.logical.php) |
| left | xor | [logical](http://php.net/manual/en/language.operators.logical.php) |
| left | or | [logical](http://php.net/manual/en/language.operators.logical.php) |
| left | , | many uses |

**Example #1 Associativity**

<?php  
$a = 3 \* 3 % 5; // (3 \* 3) % 5 = 4  
// ternary operator associativity differs from C/C++  
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2  
  
$a = 1;  
$b = 2;  
$a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5  
?>

Operator precedence and associativity only determine how expressions are grouped, they do not specify an order of evaluation. PHP does not (in the general case) specify in which order an expression is evaluated and code that assumes a specific order of evaluation should be avoided, because the behavior can change between versions of PHP or depending on the surrounding code.

**Example #2 Undefined order of evaluation**

<?php  
$a = 1;  
echo $a + $a++; // may print either 2 or 3  
  
$i = 1;  
$array[$i] = $i++; // may set either index 1 or 2  
?>

**Note**:

Although = has a lower precedence than most other operators, PHP will still allow expressions similar to the following: if (!$a = foo()), in which case the return value of foo() is put into $a.

## Arithmetic Operators [¶](http://php.net/manual/en/language.operators.arithmetic.php#language.operators.arithmetic)

Remember basic arithmetic from school? These work just like those.

| **Arithmetic Operators** | | |
| --- | --- | --- |
| **Example** | **Name** | **Result** |
| -$a | Negation | Opposite of $a. |
| $a + $b | Addition | Sum of $a and $b. |
| $a - $b | Subtraction | Difference of $a and $b. |
| $a \* $b | Multiplication | Product of $a and $b. |
| $a / $b | Division | Quotient of $a and $b. |
| $a % $b | Modulus | Remainder of $a divided by $b. |
| $a \*\* $b | Exponentiation | Result of raising $a to the $b'th power. Introduced in PHP 5.6. |

The division operator ("/") returns a float value unless the two operands are integers (or strings that get converted to integers) and the numbers are evenly divisible, in which case an integer value will be returned.

Operands of modulus are converted to integers (by stripping the decimal part) before processing.

The result of the modulus operator % has the same sign as the dividend — that is, the result of $a % $b will have the same sign as $a. For example:

<?php  
  
echo (5 % 3)."\n";           // prints 2  
echo (5 % -3)."\n";          // prints 2  
echo (-5 % 3)."\n";          // prints -2  
echo (-5 % -3)."\n";         // prints -2  
  
?>

* [abs](http://php.net/manual/en/function.abs.php) — Absolute value
* [acos](http://php.net/manual/en/function.acos.php) — Arc cosine
* [acosh](http://php.net/manual/en/function.acosh.php) — Inverse hyperbolic cosine
* [asin](http://php.net/manual/en/function.asin.php) — Arc sine
* [asinh](http://php.net/manual/en/function.asinh.php) — Inverse hyperbolic sine
* [atan2](http://php.net/manual/en/function.atan2.php) — Arc tangent of two variables
* [atan](http://php.net/manual/en/function.atan.php) — Arc tangent
* [atanh](http://php.net/manual/en/function.atanh.php) — Inverse hyperbolic tangent
* [base\_convert](http://php.net/manual/en/function.base-convert.php) — Convert a number between arbitrary bases
* [bindec](http://php.net/manual/en/function.bindec.php) — Binary to decimal
* [ceil](http://php.net/manual/en/function.ceil.php) — Round fractions up
* [cos](http://php.net/manual/en/function.cos.php) — Cosine
* [cosh](http://php.net/manual/en/function.cosh.php) — Hyperbolic cosine
* [decbin](http://php.net/manual/en/function.decbin.php) — Decimal to binary
* [dechex](http://php.net/manual/en/function.dechex.php) — Decimal to hexadecimal
* [decoct](http://php.net/manual/en/function.decoct.php) — Decimal to octal
* [deg2rad](http://php.net/manual/en/function.deg2rad.php) — Converts the number in degrees to the radian equivalent
* [exp](http://php.net/manual/en/function.exp.php) — Calculates the exponent of e
* [expm1](http://php.net/manual/en/function.expm1.php) — Returns exp(number) - 1, computed in a way that is accurate even when the value of number is close to zero
* [floor](http://php.net/manual/en/function.floor.php) — Round fractions down
* [fmod](http://php.net/manual/en/function.fmod.php) — Returns the floating point remainder (modulo) of the division of the arguments
* [getrandmax](http://php.net/manual/en/function.getrandmax.php) — Show largest possible random value
* [hexdec](http://php.net/manual/en/function.hexdec.php) — Hexadecimal to decimal
* [hypot](http://php.net/manual/en/function.hypot.php) — Calculate the length of the hypotenuse of a right-angle triangle
* [is\_finite](http://php.net/manual/en/function.is-finite.php) — Finds whether a value is a legal finite number
* [is\_infinite](http://php.net/manual/en/function.is-infinite.php) — Finds whether a value is infinite
* [is\_nan](http://php.net/manual/en/function.is-nan.php) — Finds whether a value is not a number
* [lcg\_value](http://php.net/manual/en/function.lcg-value.php) — Combined linear congruential generator
* [log10](http://php.net/manual/en/function.log10.php) — Base-10 logarithm
* [log1p](http://php.net/manual/en/function.log1p.php) — Returns log(1 + number), computed in a way that is accurate even when the value of number is close to zero
* [log](http://php.net/manual/en/function.log.php) — Natural logarithm
* [max](http://php.net/manual/en/function.max.php) — Find highest value
* [min](http://php.net/manual/en/function.min.php) — Find lowest value
* [mt\_getrandmax](http://php.net/manual/en/function.mt-getrandmax.php) — Show largest possible random value
* [mt\_rand](http://php.net/manual/en/function.mt-rand.php) — Generate a better random value
* [mt\_srand](http://php.net/manual/en/function.mt-srand.php) — Seed the better random number generator
* [octdec](http://php.net/manual/en/function.octdec.php) — Octal to decimal
* [pi](http://php.net/manual/en/function.pi.php) — Get value of pi
* [pow](http://php.net/manual/en/function.pow.php) — Exponential expression
* [rad2deg](http://php.net/manual/en/function.rad2deg.php) — Converts the radian number to the equivalent number in degrees
* [rand](http://php.net/manual/en/function.rand.php) — Generate a random integer
* [round](http://php.net/manual/en/function.round.php) — Rounds a float
* [sin](http://php.net/manual/en/function.sin.php) — Sine
* [sinh](http://php.net/manual/en/function.sinh.php) — Hyperbolic sine
* [sqrt](http://php.net/manual/en/function.sqrt.php) — Square root
* [srand](http://php.net/manual/en/function.srand.php) — Seed the random number generator
* [tan](http://php.net/manual/en/function.tan.php) — Tangent
* [tanh](http://php.net/manual/en/function.tanh.php) — Hyperbolic tangent

## Assignment Operators [¶](http://php.net/manual/en/language.operators.assignment.php#language.operators.assignment)

The basic assignment operator is "=". Your first inclination might be to think of this as "equal to". Don't. It really means that the left operand gets set to the value of the expression on the right (that is, "gets set to").

The value of an assignment expression is the value assigned. That is, the value of "$a = 3" is 3. This allows you to do some tricky things:

<?php  
  
$a = ($b = 4) + 5; // $a is equal to 9 now, and $b has been set to 4.  
  
?>

For arrays, assigning a value to a named key is performed using the "=>" operator. The [precedence](http://php.net/manual/en/language.operators.precedence.php) of this operator is the same as other assignment operators.

In addition to the basic assignment operator, there are "combined operators" for all of the [binary arithmetic](http://php.net/manual/en/language.operators.php), array union and string operators that allow you to use a value in an expression and then set its value to the result of that expression. For example:

<?php  
  
$a = 3;  
$a += 5; // sets $a to 8, as if we had said: $a = $a + 5;  
$b = "Hello ";  
$b .= "There!"; // sets $b to "Hello There!", just like $b = $b . "There!";  
  
?>

Note that the assignment copies the original variable to the new one (assignment by value), so changes to one will not affect the other. This may also have relevance if you need to copy something like a large array inside a tight loop.

An exception to the usual assignment by value behaviour within PHP occurs with [object](http://php.net/manual/en/language.types.object.php)s, which are assigned by reference in PHP 5. Objects may be explicitly copied via the [clone](http://php.net/manual/en/language.oop5.cloning.php) keyword.

### Assignment by Reference [¶](http://php.net/manual/en/language.operators.assignment.php#language.operators.assignment.reference)

Assignment by reference is also supported, using the "$var = &$othervar;" syntax. Assignment by reference means that both variables end up pointing at the same data, and nothing is copied anywhere.

**Example #1 Assigning by reference**

<?php  
$a = 3;  
$b = &$a; // $b is a reference to $a  
  
print "$a\n"; // prints 3  
print "$b\n"; // prints 3  
  
$a = 4; // change $a  
  
print "$a\n"; // prints 4  
print "$b\n"; // prints 4 as well, since $b is a reference to $a, which has  
              // been changed  
?>

As of PHP 5, the [new](http://php.net/manual/en/language.oop5.basic.php#language.oop5.basic.new) operator returns a reference automatically, so assigning the result of [new](http://php.net/manual/en/language.oop5.basic.php#language.oop5.basic.new) by reference results in an **E\_DEPRECATED** message in PHP 5.3 and later, and an **E\_STRICT** message in earlier versions.

For example, this code will result in a warning:

<?php  
class C {}  
  
/\* The following line generates the following error message:  
 \* Deprecated: Assigning the return value of new by reference is deprecated in...  
 \*/  
$o = &new C;  
?>

## Comparison Operators [¶](http://php.net/manual/en/language.operators.comparison.php#language.operators.comparison)

Comparison operators, as their name implies, allow you to compare two values. You may also be interested in viewing [the type comparison tables](http://php.net/manual/en/types.comparisons.php), as they show examples of various type related comparisons.

| **Comparison Operators** | | |
| --- | --- | --- |
| **Example** | **Name** | **Result** |
| $a == $b | Equal | **TRUE** if $a is equal to $b after type juggling. |
| $a === $b | Identical | **TRUE** if $a is equal to $b, and they are of the same type. |
| $a != $b | Not equal | **TRUE** if $a is not equal to $b after type juggling. |
| $a <> $b | Not equal | **TRUE** if $a is not equal to $b after type juggling. |
| $a !== $b | Not identical | **TRUE** if $a is not equal to $b, or they are not of the same type. |
| $a < $b | Less than | **TRUE** if $a is strictly less than $b. |
| $a > $b | Greater than | **TRUE** if $a is strictly greater than $b. |
| $a <= $b | Less than or equal to | **TRUE** if $a is less than or equal to $b. |
| $a >= $b | Greater than or equal to | **TRUE** if $a is greater than or equal to $b. |

If you compare a number with a string or the comparison involves numerical strings, then each string is [converted to a number](http://php.net/manual/en/language.types.string.php#language.types.string.conversion) and the comparison performed numerically. These rules also apply to the [switch](http://php.net/manual/en/control-structures.switch.php) statement. The type conversion does not take place when the comparison is === or !== as this involves comparing the type as well as the value.

<?php  
var\_dump(0 == "a"); // 0 == 0 -> true  
var\_dump("1" == "01"); // 1 == 1 -> true  
var\_dump("10" == "1e1"); // 10 == 10 -> true  
var\_dump(100 == "1e2"); // 100 == 100 -> true  
  
switch ("a") {  
case 0:  
    echo "0";  
    break;  
case "a": // never reached because "a" is already matched with 0  
    echo "a";  
    break;  
}  
?>

For various types, comparison is done according to the following table (in order).

| **Comparison with Various Types** | | |
| --- | --- | --- |
| **Type of Operand 1** | **Type of Operand 2** | **Result** |
| [null](http://php.net/manual/en/language.types.null.php) or [string](http://php.net/manual/en/language.types.string.php) | [string](http://php.net/manual/en/language.types.string.php) | Convert **NULL** to "", numerical or lexical comparison |
| [bool](http://php.net/manual/en/language.types.boolean.php) or [null](http://php.net/manual/en/language.types.null.php) | anything | Convert both sides to [bool](http://php.net/manual/en/language.types.boolean.php), **FALSE** < **TRUE** |
| [object](http://php.net/manual/en/language.types.object.php) | [object](http://php.net/manual/en/language.types.object.php) | Built-in classes can define its own comparison, different classes are uncomparable, same class - compare properties the same way as arrays (PHP 4), PHP 5 has its own [explanation](http://php.net/manual/en/language.oop5.object-comparison.php) |
| [string](http://php.net/manual/en/language.types.string.php), [resource](http://php.net/manual/en/language.types.resource.php) or [number](http://php.net/manual/en/language.pseudo-types.php#language.types.number) | [string](http://php.net/manual/en/language.types.string.php), [resource](http://php.net/manual/en/language.types.resource.php) or [number](http://php.net/manual/en/language.pseudo-types.php#language.types.number) | Translate strings and resources to numbers, usual math |
| [array](http://php.net/manual/en/language.types.array.php) | [array](http://php.net/manual/en/language.types.array.php) | Array with fewer members is smaller, if key from operand 1 is not found in operand 2 then arrays are uncomparable, otherwise - compare value by value (see following example) |
| [object](http://php.net/manual/en/language.types.object.php) | anything | [object](http://php.net/manual/en/language.types.object.php) is always greater |
| [array](http://php.net/manual/en/language.types.array.php) | anything | [array](http://php.net/manual/en/language.types.array.php) is always greater |

**Example #1 Boolean/null comparison**

<?php  
// Bool and null are compared as bool always  
var\_dump(1 == TRUE);  // TRUE - same as (bool)1 == TRUE  
var\_dump(0 == FALSE); // TRUE - same as (bool)0 == FALSE  
var\_dump(100 < TRUE); // FALSE - same as (bool)100 < TRUE  
var\_dump(-10 < FALSE);// FALSE - same as (bool)-10 < FALSE  
var\_dump(min(-100, -10, NULL, 10, 100)); // NULL - (bool)NULL < (bool)-100 is FALSE < TRUE  
?>

**Example #2 Transcription of standard array comparison**

<?php  
// Arrays are compared like this with standard comparison operators  
function standard\_array\_compare($op1, $op2)  
{  
    if (count($op1) < count($op2)) {  
        return -1; // $op1 < $op2  
    } elseif (count($op1) > count($op2)) {  
        return 1; // $op1 > $op2  
    }  
    foreach ($op1 as $key => $val) {  
        if (!array\_key\_exists($key, $op2)) {  
            return null; // uncomparable  
        } elseif ($val < $op2[$key]) {  
            return -1;  
        } elseif ($val > $op2[$key]) {  
            return 1;  
        }  
    }  
    return 0; // $op1 == $op2  
}  
?>

See also [strcasecmp()](http://php.net/manual/en/function.strcasecmp.php), [strcmp()](http://php.net/manual/en/function.strcmp.php), [Array operators](http://php.net/manual/en/language.operators.array.php), and the manual section on [Types](http://php.net/manual/en/language.types.php).

**Warning**

# Comparison of floating point numbers

Because of the way [float](http://php.net/manual/en/language.types.float.php)s are represented internally, you should not test two [float](http://php.net/manual/en/language.types.float.php)s for equality.

See the documentation for [float](http://php.net/manual/en/language.types.float.php) for more information.

### Ternary Operator [¶](http://php.net/manual/en/language.operators.comparison.php#language.operators.comparison.ternary)

Another conditional operator is the "?:" (or ternary) operator.

**Example #3 Assigning a default value**

<?php  
// Example usage for: Ternary Operator  
$action = (empty($\_POST['action'])) ? 'default' : $\_POST['action'];  
  
// The above is identical to this if/else statement  
if (empty($\_POST['action'])) {  
    $action = 'default';  
} else {  
    $action = $\_POST['action'];  
}  
  
?>

The expression (expr1) ? (expr2) : (expr3) evaluates to expr2 if expr1 evaluates to **TRUE**, and expr3 if expr1 evaluates to **FALSE**.

Since PHP 5.3, it is possible to leave out the middle part of the ternary operator. Expression expr1 ?: expr3 returns expr1 if expr1 evaluates to **TRUE**, and expr3 otherwise.

**Note**: Please note that the ternary operator is an expression, and that it doesn't evaluate to a variable, but to the result of an expression. This is important to know if you want to return a variable by reference. The statement return $var == 42 ? $a : $b; in a return-by-reference function will therefore not work and a warning is issued in later PHP versions.

**Note**:

It is recommended that you avoid "stacking" ternary expressions. PHP's behaviour when using more than one ternary operator within a single statement is non-obvious:

**Example #4 Non-obvious Ternary Behaviour**

<?php  
// on first glance, the following appears to output 'true'  
echo (true?'true':false?'t':'f');  
  
// however, the actual output of the above is 't'  
// this is because ternary expressions are evaluated from left to right  
  
// the following is a more obvious version of the same code as above  
echo ((true ? 'true' : false) ? 't' : 'f');  
  
// here, you can see that the first expression is evaluated to 'true', which  
// in turn evaluates to (bool)true, thus returning the true branch of the  
// second ternary expression.  
?>

## Error Control Operators [¶](http://php.net/manual/en/language.operators.errorcontrol.php#language.operators.errorcontrol)

PHP supports one error control operator: the at sign (@). When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored.

If you have set a custom error handler function with [set\_error\_handler()](http://php.net/manual/en/function.set-error-handler.php) then it will still get called, but this custom error handler can (and should) call [error\_reporting()](http://php.net/manual/en/function.error-reporting.php) which will return 0 when the call that triggered the error was preceded by an @.

If the [**track\_errors**](http://php.net/manual/en/errorfunc.configuration.php#ini.track-errors) feature is enabled, any error message generated by the expression will be saved in the variable [*$php\_errormsg*](http://php.net/manual/en/reserved.variables.phperrormsg.php). This variable will be overwritten on each error, so check early if you want to use it.

<?php  
/\* Intentional file error \*/  
$my\_file = @file ('non\_existent\_file') or  
    die ("Failed opening file: error was '$php\_errormsg'");  
  
// this works for any expression, not just functions:  
$value = @$cache[$key];  
// will not issue a notice if the index $key doesn't exist.  
  
?>

**Note**: The @-operator works only on [expressions](http://php.net/manual/en/language.expressions.php). A simple rule of thumb is: if you can take the value of something, you can prepend the @ operator to it. For instance, you can prepend it to variables, function and [include](http://php.net/manual/en/function.include.php) calls, constants, and so forth. You cannot prepend it to function or class definitions, or conditional structures such as if and [foreach](http://php.net/manual/en/control-structures.foreach.php), and so forth.

See also [error\_reporting()](http://php.net/manual/en/function.error-reporting.php) and the manual section for [Error Handling and Logging functions](http://php.net/manual/en/ref.errorfunc.php).

**Warning**

Currently the "@" error-control operator prefix will even disable error reporting for critical errors that will terminate script execution. Among other things, this means that if you use "@" to suppress errors from a certain function and either it isn't available or has been mistyped, the script will die right there with no indication as to why.

## Execution Operators [¶](http://php.net/manual/en/language.operators.execution.php#language.operators.execution)

PHP supports one execution operator: backticks (``). Note that these are not single-quotes! PHP will attempt to execute the contents of the backticks as a shell command; the output will be returned (i.e., it won't simply be dumped to output; it can be assigned to a variable). Use of the backtick operator is identical to [shell\_exec()](http://php.net/manual/en/function.shell-exec.php).

<?php  
$output = `ls -al`;  
echo "<pre>$output</pre>";  
?>

**Note**:

The backtick operator is disabled when [safe mode](http://php.net/manual/en/ini.sect.safe-mode.php#ini.safe-mode) is enabled or [shell\_exec()](http://php.net/manual/en/function.shell-exec.php) is disabled.

**Note**:

Unlike some other languages, backticks cannot be used within double-quoted strings.

See also the manual section on [Program Execution functions](http://php.net/manual/en/ref.exec.php), [popen()](http://php.net/manual/en/function.popen.php) [proc\_open()](http://php.net/manual/en/function.proc-open.php), and [Using PHP from the commandline](http://php.net/manual/en/features.commandline.php).

## Incrementing/Decrementing Operators ¶

PHP supports C-style pre- and post-increment and decrement operators.

**Note**: The increment/decrement operators only affect numbers and strings. Arrays, objects and resources are not affected. Decrementing **NULL** values has no effect too, but incrementing them results in 1.

| **Increment/decrement Operators** | | |
| --- | --- | --- |
| **Example** | **Name** | **Effect** |
| ++$a | Pre-increment | Increments $a by one, then returns $a. |
| $a++ | Post-increment | Returns $a, then increments $a by one. |
| --$a | Pre-decrement | Decrements $a by one, then returns $a. |
| $a-- | Post-decrement | Returns $a, then decrements $a by one. |

Here's a simple example script:

<?php  
echo "<h3>Postincrement</h3>";  
$a = 5;  
echo "Should be 5: " . $a++ . "<br />\n";  
echo "Should be 6: " . $a . "<br />\n";  
  
echo "<h3>Preincrement</h3>";  
$a = 5;  
echo "Should be 6: " . ++$a . "<br />\n";  
echo "Should be 6: " . $a . "<br />\n";  
  
echo "<h3>Postdecrement</h3>";  
$a = 5;  
echo "Should be 5: " . $a-- . "<br />\n";  
echo "Should be 4: " . $a . "<br />\n";  
  
echo "<h3>Predecrement</h3>";  
$a = 5;  
echo "Should be 4: " . --$a . "<br />\n";  
echo "Should be 4: " . $a . "<br />\n";  
?>

PHP follows Perl's convention when dealing with arithmetic operations on character variables and not C's. For example, in PHP and Perl $a = 'Z'; $a++; turns $a into 'AA', while in C a = 'Z'; a++; turns a into '[' (ASCII value of 'Z' is 90, ASCII value of '[' is 91). Note that character variables can be incremented but not decremented and even so only plain ASCII alphabets and digits (a-z, A-Z and 0-9) are supported. Incrementing/decrementing other character variables has no effect, the original string is unchanged.

**Example #1 Arithmetic Operations on Character Variables**

<?php  
echo '== Alphabets ==' . PHP\_EOL;  
$s = 'W';  
for ($n=0; $n<6; $n++) {  
    echo ++$s . PHP\_EOL;  
}  
// Digit characters behave differently  
echo '== Digits ==' . PHP\_EOL;  
$d = 'A8';  
for ($n=0; $n<6; $n++) {  
    echo ++$d . PHP\_EOL;  
}  
$d = 'A08';  
for ($n=0; $n<6; $n++) {  
    echo ++$d . PHP\_EOL;  
}  
?>

The above example will output:

== Characters ==

X

Y

Z

AA

AB

AC

== Digits ==

A9

B0

B1

B2

B3

B4

A09

A10

A11

A12

A13

A14

Incrementing or decrementing booleans has no effect.

## Logical Operators [¶](http://php.net/manual/en/language.operators.logical.php#language.operators.logical)

| **Logical Operators** | | |
| --- | --- | --- |
| **Example** | **Name** | **Result** |
| $a and $b | And | **TRUE** if both $a and $b are **TRUE**. |
| $a or $b | Or | **TRUE** if either $a or $b is **TRUE**. |
| $a xor $b | Xor | **TRUE** if either $a or $b is **TRUE**, but not both. |
| ! $a | Not | **TRUE** if $a is not **TRUE**. |
| $a && $b | And | **TRUE** if both $a and $b are **TRUE**. |
| $a || $b | Or | **TRUE** if either $a or $b is **TRUE**. |

The reason for the two different variations of "and" and "or" operators is that they operate at different precedences. (See [Operator Precedence](http://php.net/manual/en/language.operators.precedence.php).)

**Example #1 Logical operators illustrated**

<?php  
  
// --------------------  
// foo() will never get called as those operators are short-circuit  
  
$a = (false && foo());  
$b = (true  || foo());  
$c = (false and foo());  
$d = (true  or  foo());  
  
// --------------------  
// "||" has a greater precedence than "or"  
  
// The result of the expression (false || true) is assigned to $e  
// Acts like: ($e = (false || true))  
$e = false || true;  
  
// The constant false is assigned to $f and then true is ignored  
// Acts like: (($f = false) or true)  
$f = false or true;  
  
var\_dump($e, $f);  
  
// --------------------  
// "&&" has a greater precedence than "and"  
  
// The result of the expression (true && false) is assigned to $g  
// Acts like: ($g = (true && false))  
$g = true && false;  
  
// The constant true is assigned to $h and then false is ignored  
// Acts like: (($h = true) and false)  
$h = true and false;  
  
var\_dump($g, $h);  
?>

The above example will output something similar to:

bool(true)

bool(false)

bool(false)

bool(true)

## String Operators [¶](http://php.net/manual/en/language.operators.string.php#language.operators.string)

There are two [string](http://php.net/manual/en/language.types.string.php) operators. The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side. Please read [Assignment Operators](http://php.net/manual/en/language.operators.assignment.php) for more information.

<?php  
$a = "Hello ";  
$b = $a . "World!"; // now $b contains "Hello World!"  
  
$a = "Hello ";  
$a .= "World!";     // now $a contains "Hello World!"  
?>

# String Functions [¶](http://php.net/manual/en/ref.strings.php#ref.strings)

# See Also

For even more powerful string handling and manipulating functions take a look at the [POSIX regular expression functions](http://php.net/manual/en/ref.regex.php) and the [Perl compatible regular expression functions](http://php.net/manual/en/ref.pcre.php).

## Table of Contents [¶](http://php.net/manual/en/ref.strings.php#ref.strings)

* [addcslashes](http://php.net/manual/en/function.addcslashes.php) — Quote string with slashes in a C style
* [addslashes](http://php.net/manual/en/function.addslashes.php) — Quote string with slashes
* [bin2hex](http://php.net/manual/en/function.bin2hex.php) — Convert binary data into hexadecimal representation
* [chop](http://php.net/manual/en/function.chop.php) — Alias of rtrim
* [chr](http://php.net/manual/en/function.chr.php) — Return a specific character
* [chunk\_split](http://php.net/manual/en/function.chunk-split.php) — Split a string into smaller chunks
* [convert\_cyr\_string](http://php.net/manual/en/function.convert-cyr-string.php) — Convert from one Cyrillic character set to another
* [convert\_uudecode](http://php.net/manual/en/function.convert-uudecode.php) — Decode a uuencoded string
* [convert\_uuencode](http://php.net/manual/en/function.convert-uuencode.php) — Uuencode a string
* [count\_chars](http://php.net/manual/en/function.count-chars.php) — Return information about characters used in a string
* [crc32](http://php.net/manual/en/function.crc32.php) — Calculates the crc32 polynomial of a string
* [crypt](http://php.net/manual/en/function.crypt.php) — One-way string hashing
* [echo](http://php.net/manual/en/function.echo.php) — Output one or more strings
* [explode](http://php.net/manual/en/function.explode.php) — Split a string by string
* [fprintf](http://php.net/manual/en/function.fprintf.php) — Write a formatted string to a stream
* [get\_html\_translation\_table](http://php.net/manual/en/function.get-html-translation-table.php) — Returns the translation table used by htmlspecialchars and htmlentities
* [hebrev](http://php.net/manual/en/function.hebrev.php) — Convert logical Hebrew text to visual text
* [hebrevc](http://php.net/manual/en/function.hebrevc.php) — Convert logical Hebrew text to visual text with newline conversion
* [hex2bin](http://php.net/manual/en/function.hex2bin.php) — Decodes a hexadecimally encoded binary string
* [html\_entity\_decode](http://php.net/manual/en/function.html-entity-decode.php) — Convert all HTML entities to their applicable characters
* [htmlentities](http://php.net/manual/en/function.htmlentities.php) — Convert all applicable characters to HTML entities
* [htmlspecialchars\_decode](http://php.net/manual/en/function.htmlspecialchars-decode.php) — Convert special HTML entities back to characters
* [htmlspecialchars](http://php.net/manual/en/function.htmlspecialchars.php) — Convert special characters to HTML entities
* [implode](http://php.net/manual/en/function.implode.php) — Join array elements with a string
* [join](http://php.net/manual/en/function.join.php) — Alias of implode
* [lcfirst](http://php.net/manual/en/function.lcfirst.php) — Make a string's first character lowercase
* [levenshtein](http://php.net/manual/en/function.levenshtein.php) — Calculate Levenshtein distance between two strings
* [localeconv](http://php.net/manual/en/function.localeconv.php) — Get numeric formatting information
* [ltrim](http://php.net/manual/en/function.ltrim.php) — Strip whitespace (or other characters) from the beginning of a string
* [md5\_file](http://php.net/manual/en/function.md5-file.php) — Calculates the md5 hash of a given file
* [md5](http://php.net/manual/en/function.md5.php) — Calculate the md5 hash of a string
* [metaphone](http://php.net/manual/en/function.metaphone.php) — Calculate the metaphone key of a string
* [money\_format](http://php.net/manual/en/function.money-format.php) — Formats a number as a currency string
* [nl\_langinfo](http://php.net/manual/en/function.nl-langinfo.php) — Query language and locale information
* [nl2br](http://php.net/manual/en/function.nl2br.php) — Inserts HTML line breaks before all newlines in a string
* [number\_format](http://php.net/manual/en/function.number-format.php) — Format a number with grouped thousands
* [ord](http://php.net/manual/en/function.ord.php) — Return ASCII value of character
* [parse\_str](http://php.net/manual/en/function.parse-str.php) — Parses the string into variables
* [print](http://php.net/manual/en/function.print.php) — Output a string
* [printf](http://php.net/manual/en/function.printf.php) — Output a formatted string
* [quoted\_printable\_decode](http://php.net/manual/en/function.quoted-printable-decode.php) — Convert a quoted-printable string to an 8 bit string
* [quoted\_printable\_encode](http://php.net/manual/en/function.quoted-printable-encode.php) — Convert a 8 bit string to a quoted-printable string
* [quotemeta](http://php.net/manual/en/function.quotemeta.php) — Quote meta characters
* [rtrim](http://php.net/manual/en/function.rtrim.php) — Strip whitespace (or other characters) from the end of a string
* [setlocale](http://php.net/manual/en/function.setlocale.php) — Set locale information
* [sha1\_file](http://php.net/manual/en/function.sha1-file.php) — Calculate the sha1 hash of a file
* [sha1](http://php.net/manual/en/function.sha1.php) — Calculate the sha1 hash of a string
* [similar\_text](http://php.net/manual/en/function.similar-text.php) — Calculate the similarity between two strings
* [soundex](http://php.net/manual/en/function.soundex.php) — Calculate the soundex key of a string
* [sprintf](http://php.net/manual/en/function.sprintf.php) — Return a formatted string
* [sscanf](http://php.net/manual/en/function.sscanf.php) — Parses input from a string according to a format
* [str\_getcsv](http://php.net/manual/en/function.str-getcsv.php) — Parse a CSV string into an array
* [str\_ireplace](http://php.net/manual/en/function.str-ireplace.php) — Case-insensitive version of str\_replace.
* [str\_pad](http://php.net/manual/en/function.str-pad.php) — Pad a string to a certain length with another string
* [str\_repeat](http://php.net/manual/en/function.str-repeat.php) — Repeat a string
* [str\_replace](http://php.net/manual/en/function.str-replace.php) — Replace all occurrences of the search string with the replacement string
* [str\_rot13](http://php.net/manual/en/function.str-rot13.php) — Perform the rot13 transform on a string
* [str\_shuffle](http://php.net/manual/en/function.str-shuffle.php) — Randomly shuffles a string
* [str\_split](http://php.net/manual/en/function.str-split.php) — Convert a string to an array
* [str\_word\_count](http://php.net/manual/en/function.str-word-count.php) — Return information about words used in a string
* [strcasecmp](http://php.net/manual/en/function.strcasecmp.php) — Binary safe case-insensitive string comparison
* [strchr](http://php.net/manual/en/function.strchr.php) — Alias of strstr
* [strcmp](http://php.net/manual/en/function.strcmp.php) — Binary safe string comparison
* [strcoll](http://php.net/manual/en/function.strcoll.php) — Locale based string comparison
* [strcspn](http://php.net/manual/en/function.strcspn.php) — Find length of initial segment not matching mask
* [strip\_tags](http://php.net/manual/en/function.strip-tags.php) — Strip HTML and PHP tags from a string
* [stripcslashes](http://php.net/manual/en/function.stripcslashes.php) — Un-quote string quoted with addcslashes
* [stripos](http://php.net/manual/en/function.stripos.php) — Find the position of the first occurrence of a case-insensitive substring in a string
* [stripslashes](http://php.net/manual/en/function.stripslashes.php) — Un-quotes a quoted string
* [stristr](http://php.net/manual/en/function.stristr.php) — Case-insensitive strstr
* [strlen](http://php.net/manual/en/function.strlen.php) — Get string length
* [strnatcasecmp](http://php.net/manual/en/function.strnatcasecmp.php) — Case insensitive string comparisons using a "natural order" algorithm
* [strnatcmp](http://php.net/manual/en/function.strnatcmp.php) — String comparisons using a "natural order" algorithm
* [strncasecmp](http://php.net/manual/en/function.strncasecmp.php) — Binary safe case-insensitive string comparison of the first n characters
* [strncmp](http://php.net/manual/en/function.strncmp.php) — Binary safe string comparison of the first n characters
* [strpbrk](http://php.net/manual/en/function.strpbrk.php) — Search a string for any of a set of characters
* [strpos](http://php.net/manual/en/function.strpos.php) — Find the position of the first occurrence of a substring in a string
* [strrchr](http://php.net/manual/en/function.strrchr.php) — Find the last occurrence of a character in a string
* [strrev](http://php.net/manual/en/function.strrev.php) — Reverse a string
* [strripos](http://php.net/manual/en/function.strripos.php) — Find the position of the last occurrence of a case-insensitive substring in a string
* [strrpos](http://php.net/manual/en/function.strrpos.php) — Find the position of the last occurrence of a substring in a string
* [strspn](http://php.net/manual/en/function.strspn.php) — Finds the length of the initial segment of a string consisting entirely of characters contained within a given mask.
* [strstr](http://php.net/manual/en/function.strstr.php) — Find the first occurrence of a string
* [strtok](http://php.net/manual/en/function.strtok.php) — Tokenize string
* [strtolower](http://php.net/manual/en/function.strtolower.php) — Make a string lowercase
* [strtoupper](http://php.net/manual/en/function.strtoupper.php) — Make a string uppercase
* [strtr](http://php.net/manual/en/function.strtr.php) — Translate characters or replace substrings
* [substr\_compare](http://php.net/manual/en/function.substr-compare.php) — Binary safe comparison of two strings from an offset, up to length characters
* [substr\_count](http://php.net/manual/en/function.substr-count.php) — Count the number of substring occurrences
* [substr\_replace](http://php.net/manual/en/function.substr-replace.php) — Replace text within a portion of a string
* [substr](http://php.net/manual/en/function.substr.php) — Return part of a string
* [trim](http://php.net/manual/en/function.trim.php) — Strip whitespace (or other characters) from the beginning and end of a string
* [ucfirst](http://php.net/manual/en/function.ucfirst.php) — Make a string's first character uppercase
* [ucwords](http://php.net/manual/en/function.ucwords.php) — Uppercase the first character of each word in a string
* [vfprintf](http://php.net/manual/en/function.vfprintf.php) — Write a formatted string to a stream
* [vprintf](http://php.net/manual/en/function.vprintf.php) — Output a formatted string
* [vsprintf](http://php.net/manual/en/function.vsprintf.php) — Return a formatted string
* [wordwrap](http://php.net/manual/en/function.wordwrap.php) — Wraps a string to a given number of characters

## Array Operators [¶](http://php.net/manual/en/language.operators.array.php#language.operators.array)

| **Array Operators** | | |
| --- | --- | --- |
| **Example** | **Name** | **Result** |
| $a + $b | Union | Union of $a and $b. |
| $a == $b | Equality | **TRUE** if $a and $b have the same key/value pairs. |
| $a === $b | Identity | **TRUE** if $a and $b have the same key/value pairs in the same order and of the same types. |
| $a != $b | Inequality | **TRUE** if $a is not equal to $b. |
| $a <> $b | Inequality | **TRUE** if $a is not equal to $b. |
| $a !== $b | Non-identity | **TRUE** if $a is not identical to $b. |

The + operator returns the right-hand array appended to the left-hand array; for keys that exist in both arrays, the elements from the left-hand array will be used, and the matching elements from the right-hand array will be ignored.

<?php  
$a = array("a" => "apple", "b" => "banana");  
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");  
  
$c = $a + $b; // Union of $a and $b  
echo "Union of \$a and \$b: \n";  
var\_dump($c);  
  
$c = $b + $a; // Union of $b and $a  
echo "Union of \$b and \$a: \n";  
var\_dump($c);  
?>

When executed, this script will print the following:

Union of $a and $b:

array(3) {

["a"]=>

string(5) "apple"

["b"]=>

string(6) "banana"

["c"]=>

string(6) "cherry"

}

Union of $b and $a:

array(3) {

["a"]=>

string(4) "pear"

["b"]=>

string(10) "strawberry"

["c"]=>

string(6) "cherry"

}

Elements of arrays are equal for the comparison if they have the same key and value.

**Example #1 Comparing arrays**

<?php  
$a = array("apple", "banana");  
$b = array(1 => "banana", "0" => "apple");  
  
var\_dump($a == $b); // bool(true)  
var\_dump($a === $b); // bool(false)  
?>

See also the manual sections on the [Array type](http://php.net/manual/en/language.types.array.php) and [Array functions](http://php.net/manual/en/ref.array.php).

## Type Operators [¶](http://php.net/manual/en/language.operators.type.php#language.operators.type)

instanceof is used to determine whether a PHP variable is an instantiated object of a certain [class](http://php.net/manual/en/language.oop5.basic.php#language.oop5.basic.class):

**Example #1 Using instanceof with classes**

<?php  
class MyClass  
{  
}  
  
class NotMyClass  
{  
}  
$a = new MyClass;  
  
var\_dump($a instanceof MyClass);  
var\_dump($a instanceof NotMyClass);  
?>

The above example will output:

bool(true)

bool(false)

instanceof can also be used to determine whether a variable is an instantiated object of a class that inherits from a parent class:

**Example #2 Using instanceof with inherited classes**

<?php  
class ParentClass  
{  
}  
  
class MyClass extends ParentClass  
{  
}  
  
$a = new MyClass;  
  
var\_dump($a instanceof MyClass);  
var\_dump($a instanceof ParentClass);  
?>

The above example will output:

bool(true)

bool(true)

To check if an object is not an instanceof a class, the [logical not operator](http://php.net/manual/en/language.operators.logical.php) can be used.

**Example #3 Using instanceof to check if object is not an instanceof a class**

<?php  
class MyClass  
{  
}  
  
$a = new MyClass;  
var\_dump(!($a instanceof stdClass));  
?>

The above example will output:

bool(true)

Lastly, instanceof can also be used to determine whether a variable is an instantiated object of a class that implements an [interface](http://php.net/manual/en/language.oop5.interfaces.php):

**Example #4 Using instanceof for class**

<?php  
interface MyInterface  
{  
}  
  
class MyClass implements MyInterface  
{  
}  
  
$a = new MyClass;  
  
var\_dump($a instanceof MyClass);  
var\_dump($a instanceof MyInterface);  
?>

The above example will output:

bool(true)

bool(true)

Although instanceof is usually used with a literal classname, it can also be used with another object or a string variable:

**Example #5 Using instanceof with other variables**

<?php  
interface MyInterface  
{  
}  
  
class MyClass implements MyInterface  
{  
}  
  
$a = new MyClass;  
$b = new MyClass;  
$c = 'MyClass';  
$d = 'NotMyClass';  
  
var\_dump($a instanceof $b); // $b is an object of class MyClass  
var\_dump($a instanceof $c); // $c is a string 'MyClass'  
var\_dump($a instanceof $d); // $d is a string 'NotMyClass'  
?>

The above example will output:

bool(true)

bool(true)

bool(false)

instanceof does not throw any error if the variable being tested is not an object, it simply returns **FALSE**. Constants, however, are not allowed.

**Example #6 Using instanceof to test other variables**

<?php  
$a = 1;  
$b = NULL;  
$c = imagecreate(5, 5);  
var\_dump($a instanceof stdClass); // $a is an integer  
var\_dump($b instanceof stdClass); // $b is NULL  
var\_dump($c instanceof stdClass); // $c is a resource  
var\_dump(FALSE instanceof stdClass);  
?>

The above example will output:

bool(false)

bool(false)

bool(false)

PHP Fatal error: instanceof expects an object instance, constant given

There are a few pitfalls to be aware of. Before PHP version 5.1.0, instanceof would call [\_\_autoload()](http://php.net/manual/en/function.autoload.php) if the class name did not exist. In addition, if the class was not loaded, a fatal error would occur. This can be worked around by using a dynamic class reference, or a string variable containing the class name:

**Example #7 Avoiding classname lookups and fatal errors with instanceof in PHP 5.0**

<?php  
$d = 'NotMyClass';  
var\_dump($a instanceof $d); // no fatal error here  
?>

The above example will output:

bool(false)

The instanceof operator was introduced in PHP 5. Before this time [is\_a()](http://php.net/manual/en/function.is-a.php) was used but [is\_a()](http://php.net/manual/en/function.is-a.php) has since been deprecated in favor of instanceof. Note that as of PHP 5.3.0, [is\_a()](http://php.net/manual/en/function.is-a.php) is no longer deprecated.

See also [get\_class()](http://php.net/manual/en/function.get-class.php) and [is\_a()](http://php.net/manual/en/function.is-a.php).

**Next Topics**

## Control Structures

## Language Constructs and Functions

## Namespaces

## Extensions

## Config

## Performance/bytecode caching \*