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

Error Control Operators »
« Bitwise Operators

- PHP Manual
- Language Reference
- Operators

Change language: English

Edit Report a Bug

Comparison Operators ¶

Comparison operators, as their name implies, allow you to compare two values. You may also be interested in viewing the type comparison tables, 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.

Example	Name	Result
$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 and the comparison performed numerically. These rules also apply to the switch 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

Type of Operand 1	Type of Operand 2	Result
null or string	string	Convert NULL to "", numerical or lexical comparison
bool or null	anything	Convert both sides to bool, FALSE < TRUE
object	object	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
string, resource or number	string, resource or number	Translate strings and resources to numbers, usual math
array	array	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	anything	object is always greater
array	anything	array 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(), strcmp(), Array operators, and the manual section on Types.

Warning

Comparison of floating point numbers

Because of the way floats are represented internally, you should not test two floats for equality.

See the documentation for float for more information.

Ternary Operator ¶

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.
?>
```

add a note

User Contributed Notes 47 notes

up
down
29
Anonymous ¶
9 years ago

The following contrasts the trinary operator associativity in PHP and Java. The first test would work as expected in Java (evaluates left-to-right, associates right-to-left, like if stmnt), the second in PHP (evaluates and associates left-to-right)

```
<?php

echo "\n\n######---------- trinary operator associativity\n\n";

function trinaryTest($foo){

    $bar    = $foo > 20
            ? "greater than 20"
            : $foo > 10
                ? "greater than 10"
                : $foo > 5
                    ? "greater than 5"
                    : "not worthy of consideration";
    echo $foo." =>  ".$bar."\n";
}

echo "----trinaryTest\n\n";
trinaryTest(21);
trinaryTest(11);
trinaryTest(6);
trinaryTest(4);

function trinaryTestParens($foo){

    $bar    = $foo > 20
            ? "greater than 20"
            : ($foo > 10
                ? "greater than 10"
                : ($foo > 5
                    ? "greater than 5"
                    : "not worthy of consideration"));
    echo $foo." =>  ".$bar."\n";
}

echo "----trinaryTestParens\n\n";
trinaryTestParens(21);
trinaryTestParens(11);
trinaryTest(6);
trinaryTestParens(4);

?>

Output:
```

```
######---------- trinary operator associativity

----trinaryTest

21 =>  greater than 5
11 =>  greater than 5
6 =>  greater than 5
4 =>  not worthy of consideration

----trinaryTestParens

21 =>  greater than 20
11 =>  greater than 10
6 =>  greater than 5
4 =>  not worthy of consideration
```

up
down
25

arnaud at arnapou dot net ¶

4 years ago

I discover after 10 years of PHP development something awfull : even if you make a string comparison (both are strings), strings are tested like integers and leading "space" character (even \n, \r, \t) is ignored ....

I spent hours because of leading \n in a string ... it hurts my developper sensibility to see two strings beeing compared like integers and not like strings ... I use strcmp now for string comparison ... so stupid ...

Test code :

```
<?php

test("1234", "1234");
test("1234", " 1234");
test("1234", "\n1234");
test("1234", "1234 ");
test("1234", "1234\n");

function test($v1, $v2) {
    echo "<h1>[".show_cr($v1)."] vs [".show_cr($v2)."]</h1>";
    echo my_var_dump($v1)."<br />";
    echo my_var_dump($v2)."<br />";
    if($v1 == $v2) {
        echo "EQUAL !";
```

```
    }
    else {
        echo "DIFFERENT !";
    }
}

function show_cr($var) {
    return str_replace("\n", "\\n", $var);
}

function my_var_dump($var) {
    ob_start();
    var_dump($var);
    $dump = show_cr(trim(ob_get_contents()));
    ob_end_clean();
    return $dump;
}

?>

Displays this ->

[1234] vs [1234]
string(4) "1234"
string(4) "1234"
EQUAL !

[1234] vs [ 1234]
string(4) "1234"
string(5) " 1234"
EQUAL !

[1234] vs [\n1234]
string(4) "1234"
string(5) "\n1234"
EQUAL !

[1234] vs [1234 ]
string(4) "1234"
string(5) "1234 "
DIFFERENT !

[1234] vs [1234\n]
string(4) "1234"
```

```
string(5) "1234\n"
DIFFERENT !
```

up
down
19

crazy888s at hotmail dot com ¶

4 years ago

I couldn't find much info on stacking the new ternary operator, so I ran some tests:

```
<?php
echo 0 ?: 1 ?: 2 ?: 3; //1
echo 1 ?: 0 ?: 3 ?: 2; //1
echo 2 ?: 1 ?: 0 ?: 3; //2
echo 3 ?: 2 ?: 1 ?: 0; //3

echo 0 ?: 1 ?: 2 ?: 3; //1
echo 0 ?: 0 ?: 2 ?: 3; //2
echo 0 ?: 0 ?: 0 ?: 3; //3
?>
```

It works just as expected, returning the first non-false value within a group of expressions.

up
down
11

adam at caucho dot com ¶

8 years ago

Note: according to the spec, PHP's comparison operators are not transitive. For example, the following are all true in PHP5:

```
"11" < "a" < 2 < "11"
```

As a result, the outcome of sorting an array depends on the order the elements appear in the pre-sort array. The following code will dump out two arrays with *different* orderings:

```
<?php
$a = array(2,    "a",  "11", 2);
$b = array(2,    "11", "a",  2);
sort($a);
var_dump($a);
sort($b);
var_dump($b);
?>
```

This is not a bug report -- given the spec on this documentation page, what PHP does is "correct". But that may not be what was intended...

up
down
3

damien dot launay dot mail at gmail dot com ¶

1 year ago

I found a nice way to use of new "?:" operator:

```
$a = array();
$a['foo'] = 'oof';

$b = @ ($a['foo'] ?: 'No foo');
$c = @ ($a['bar'] ?: 'No bar');

var_dump($b, $c);
```

Output:

```
string(3) "oof"
string(6) "No bar"
```

No error is thrown and $c is set with correct value.

Benefit: no need to use isset.

up
down
5

thomas dot oldbury at tgohome dot com ¶

7 years ago

Be careful when using the ternary operator!

The following will not evaluate to the expected result:

```
<?php
echo "a string that has a " . (true) ? 'true' : 'false' . " condition in. ";
?>
```

Will print true.

Instead, use this:

```
<?php
```

```
echo "a string that has a " . ((true) ? 'true' : 'false') . " condition in. ";
?>
```

This will evaluate to the expected result: "a string that has a true condition in. "

I hope this helps.

up
down
7

kapoor_rajiv at hotmail dot com ¶

5 years ago

A quick way to do mysql bit comparison in php is to use the special character it stores .
e.g

```
<?php
                                        if ($AvailableRequests['OngoingService'] == '')
                                            echo '<td>Yes</td>';
                                        else
                                            echo '<td>No</td>';

?>
```

up
down
9

stepheneliotdewey at gmail [period] com ¶

7 years ago

Note that typecasting will NOT prevent the default behavior for converting two numeric strings to numbers when comparing them.

e.g.:

```
<?php
if ((string) '0123' == (string) '123')
    print 'equals';
else
    print 'doesn\'t equal';
?>
```

Still prints 'equals'

As far as I can tell the only way to avoid this is to use the identity comparison operators (=== and !==).

up
down
7

jwhiting at hampshire dot edu ¶
10 years ago

note: the behavior below is documented in the appendix K about type comparisons, but since it is somewhat buried i thought i should raise it here for people since it threw me for a loop until i figured it out completely.

just to clarify a tricky point about the == comparison operator when dealing with strings and numbers:

('some string' == 0) returns TRUE

however, ('123' == 0) returns FALSE

also note that ((int) 'some string') returns 0

and ((int) '123') returns 123

the behavior makes senes but you must be careful when comparing strings to numbers, e.g. when you're comparing a request variable which you expect to be numeric. its easy to fall into the trap of:

```
if ($_GET['myvar']==0) dosomething();
```

as this will dosomething() even when $_GET['myvar'] is 'some string' and clearly not the value 0

i was getting lazy with my types since php vars are so flexible, so be warned to pay attention to the details...

up
down
9

fernandoleal at dragoncs dot com ¶
7 years ago

If you need nested ifs on I var its important to group the if so it works.
Example:

```
<?php
//Dont Works
//Parse error: parse error, unexpected ':'
$var='<option value="1" '.$status == "1" ? 'selected="selected"' :''.'>Value 1</option>';
//Works:
$var='<option value="1" '.($status == "1" ? 'selected="selected"' :'').'>Value
1</option>';

echo $var;
```

```
?>
```

up
down
7

Cuong Huy To ¶

3 years ago

In the table "Comparison with Various Types", please move the last line about "Object" to be above the line about "Array", since Object is considered to be greater than Array (tested on 5.3.3)

(Please remove my "Anonymous" post of the same content before. You could check IP to see that I forgot to type my name)

up
down
7

Alex ¶

8 years ago

I think everybody should read carefully what "jeronimo at DELETE_THIS dot transartmedia dot com" wrote. It's a great pitfall even for seasoned programmers and should be looked upon with a great attention.
For example, comparing passwords with == may result in a very large security hole.

I would add some more to it:

The workaround is to use strcmp() or ===.

Note on ===:

While the php documentation says that, basically,
($a===$b) is the same as ($a==$b && gettype($a) == gettype($b)),
this is not true.

The difference between == and === is that === never does any type conversion. So, while, according to documentation, ("+0.1" === ".1") should return true (because both are strings and == returns true), === actually returns false (which is good).

up
down
4

hiroh2k at yahoo dot com ¶

10 years ago

if you want to use the ?: operator, you should be careful with the precedence.

Here's an example of the priority of operators:

```
<?php
echo 'Hello, ' . isset($i) ? 'my friend: ' . $username . ', how are you doing?' : 'my
guest, ' . $guestusername . ', please register';
?>
```

This make "'Hello, ' . isset($i)" the sentence to evaluate. So, if you think to mix more sentences with the ?: operator, please use always parentheses to force the proper evaluation of the sentence.

```
<?php
echo 'Hello, ' . (isset($i) ? 'my friend: ' . $username . ', how are you doing?' : 'my
guest, ' . $guestusername . ', please register');
?>
```

for general rule, if you mix ?: with other sentences, always close it with parentheses.

up
down
6

Hayley Watson ¶

7 years ago

Note that the "ternary operator" is better described as the "conditional operator". The former name merely notes that it has three arguments without saying anything about what it does. Needless to say, if PHP picked up any more ternary operators, this will be a problem.

"Conditional Operator" is actually descriptive of the semantics, and is the name historically given to it in, e.g., C.

up
down
7

toader_alexandru at yahoo dot com ¶

2 years ago

it looks that

if you check 0 against a string with == then PHP returns true:

```
php -r 'var_dump(0 == "statuses");'
-> returns TRUE
```

but not if your string has a number at the beginning:

```
php -r 'var_dump(0 == "2statuses");'
-> returns FALSE
```

from the specs I get it that it attempts a conversion - in this case the string to number.

so better use ===
as always :)

up
down
7

zak at minion dot net ¶

3 years ago

be careful when trying to concatenate the result of a ternary operator to a string

```
<?php
print '<div>'.(FALSE) ? 'TRUE [bad ternary]' : 'FALSE [bad ternary]';
print '<br><br>';
print '<div>'.((FALSE) ? 'TRUE [good ternary]' : 'FALSE [good ternary]');
?>
```

yields:

TRUE [bad ternary]

FALSE [good ternary]

this is because the ternary evaluates ' '.(FALSE) not (FALSE) - so the end result is TRUE

up
down
6

user@example ¶

10 years ago

With Nested ternary Operators you have to set the logical parentheses to get the correct result.

```
<?php
$test=true;
$test2=true;

($test) ? "TEST1 true" :  ($test2) ? "TEST2 true" : "false";
?>
```

This will output: TEST2 true;

correct:

```
<?php
```

```
$test=true;
$test2=true;

($test) ? "TEST1 true" : (($test2) ? "TEST2 true" : "false");
?>
```

Anyway don't nest them to much....!!

up
down
5

bimal at sanjaal dot com ¶

1 year ago

I came across peculiar outputs while I was attempting to debug a script

```
<?php
# Setup platform (pre conditions somewhere in a loop)
$index=1;
$tally = array();

# May work with warnings that $tally[$index] is not initialized
# Notice: Undefined offset: 1 in D:\htdocs\colors\ColorCompare\i.php on line #__
# It is an old fashioned way.
# $tally[$index] = $tally[$index] + 1;

# Does not work: Loops to attempt to change $index and values are aways unaffected
$tally[$index] = isset($tally[$index])?$tally[$index]:0+1;
$tally[$index] = isset($tally[$index])?$tally[$index]:0+1;
$tally[$index] = isset($tally[$index])?$tally[$index]:0+1;
/*
# These three lines output:
Array
(
    [1] => 1
)
*/

# Works: This is what I need/expect
# $tally[$index] = 1+(isset($tally[$index])?$tally[$index]:0);

print_r($tally);
?>
```

The second block obviously does not work what one expects.
Third part is good.

up
down
6
Anonymous ¶
5 years ago
Note: The ternary shortcut currently seems to be of no use in dealing with unexisting keys in an array, as PHP will throw an error. Take the following example.

```
<?php
$_POST['Unexisting'] = $_POST['Unexisting'] ?: false;
?>
```

PHP will throw an error that the "Unexisting" key does not exist. The @ operator does not work here to suppress this error.
up
down
4
alan dot g at nospam dot net ¶
4 years ago
a function to help settings default values, it returns its own first non-empty argument :

make your own eor combos !

```
<?php

/*
* Either Or
*
* usage:  $foo = eor(test1(),test2(),"default");
* usage:  $foo = eor($_GET['foo'], foogen(), $foo, "bar");
*/

function eor() {
    $vars = func_get_args();
     while (!empty($vars) && empty($defval))
         $defval = array_shift($vars);
     return $defval;
}



?>
```
up
down

3

bishop ¶

8 years ago

When you want to know if two arrays contain the same values, regardless of the values' order, you cannot use "==" or "===". In other words:

```
<?php
(array(1,2) == array(2,1)) === false;
?>
```

To answer that question, use:

```
<?php
function array_equal($a, $b) {
    return (is_array($a) && is_array($b) && array_diff($a, $b) === array_diff($b, $a));
}
?>
```

A related, but more strict problem, is if you need to ensure that two arrays contain the same key=>value pairs, regardless of the order of the pairs. In that case, use:

```
<?php
function array_identical($a, $b) {
    return (is_array($a) && is_array($b) && array_diff_assoc($a, $b) ===
array_diff_assoc($b, $a));
}
?>
```

Example:

```
<?php
$a = array (2, 1);
$b = array (1, 2);
// true === array_equal($a, $b);
// false === array_identical($a, $b);

$a = array ('a' => 2, 'b' => 1);
$b = array ('b' => 1, 'a' => 2);
// true === array_identical($a, $b)
// true === array_equal($a, $b)
?>
```

(See also the solution "rshawiii at yahoo dot com" posted)

up
down

5

rshawiii at yahoo dot com ¶

8 years ago

You can't just compare two arrays with the === operator
like you would think to find out if they are equal or not. This is more complicated when you have multi-dimensional arrays. Here is a recursive comparison function.

```
<?php
/**
* Compares two arrays to see if they contain the same values.  Returns TRUE or FALSE.
* usefull for determining if a record or block of data was modified (perhaps by user input)
* prior to setting a "date_last_updated" or skipping updating the db in the case of no change.
*
* @param array $a1
* @param array $a2
* @return boolean
*/
function array_compare_recursive($a1, $a2)
{
   if (!(is_array($a1) and (is_array($a2)))) { return FALSE;}

   if (!count($a1) == count($a2))
      {
       return FALSE; // arrays don't have same number of entries
      }

   foreach ($a1 as $key => $val)
   {
       if (!array_key_exists($key, $a2))
           {return FALSE; // uncomparable array keys don't match
              }
       elseif (is_array($val) and is_array($a2[$key]))  // if both entries are arrays then compare recursive
           {if (!array_compare_recursive($val,$a2[$key])) return FALSE;
           }
       elseif (!($val === $a2[$key])) // compare entries must be of same type.
           {return FALSE;
           }
   }
   return TRUE; // $a1 === $a2
}
?>
```

up
down
5
taras dot bogach at gmail dot com ¶
4 years ago
Boolean switch usege

```
<?php
class User_Exception extends Exception{}
class User{
  public function register($login,$pass,$passCheck)
    switch(false){
      case(strlen($pass) >= 5):
        throw new User_Exception("Password must be at last 5 chars length");
      case($pass == $passCheck):
        throw new User_Exception("Password is not confirmed!");
      case(strlen($login) >= 5):
        throw new User_Exception("Login must be at last 5 chars length");
      //Do other checks
      default:
        //Do registration
        return true;
    }
  }
  //...
}
?>
```

up
down
4
j-a-n at gmx dot de ¶
3 years ago
Please be careful when comparing strings with floats, especally when you are using the , as decimal.

```
<?php
var_dump($alt);
var_dump($neu);
var_dump($alt == $neu);
?>

string(9) "590217,73"
float(590217,73)
bool(false)
```

not the float is cast to a string and then string-compared, but the string is cast to a float and then float-compared.

to compare as strings use strval!

```
<?php
var_dump(strval($alt));
var_dump(strval($neu));
var_dump(strval($alt) == strval($neu));
?>

string(9) "590217,73"
string(9) "590217,73"
bool(true)
```

up
down
4
mail at markuszeller dot com ¶
4 years ago

I prefer writing (!$a == 'hello') much more than ($a != 'hello'), but I wondered about the performance.

So I did a benchmark:

```
<?php
for($bench = 0; $bench < 3; $bench++)
{
    $start = microtime(true);
    $a = 1;
    for($i = 0; $i < 100000000; $i++)
  {
        if(!$a == 'hello') $b++;
    }
    $end = microtime(true);
    echo "Used time: " . ($end-$start) . "\n";
}
?>
```

and it results with

```
# if($a != 'hello')
Used time: 12.552895069122
Used time: 12.548940896988
Used time: 12.470285177231
```

```
# if(!$a == 'hello')
Used time: 7.6532161235809
Used time: 7.6426539421082
Used time: 7.6452689170837
```

up
down
1

jeronimo at DELETE_THIS dot transartmedia dot com ¶
10 years ago

For converted Perl programmers: use strict comparison operators (===, !==) in place of string comparison operators (eq, ne). Don't use the simple equality operators (==, !=), because ($a == $b) will return TRUE in many situations where ($a eq $b) would return FALSE.

For instance...
"mary" == "fred" is FALSE, but
"+010" == "10.0" is TRUE (!)

In the following examples, none of the strings being compared are identical, but because PHP *can* evaluate them as numbers, it does so, and therefore finds them equal...

```
<?php

echo ("007" == "7" ? "EQUAL" : "not equal");
// Prints: EQUAL

// Surrounding the strings with single quotes (') instead of double
// quotes (") to ensure the contents aren't evaluated, and forcing
// string types has no effect.
echo ( (string)'0001' == (string)'+1.' ? "EQUAL" : "not equal");
// Prints: EQUAL

// Including non-digit characters (like leading spaces, "e", the plus
// or minus sign, period, ...) can still result in this behavior, if
// a string happens to be valid scientific notation.
echo ('  131e-2' == '001.3100' ? "EQUAL" : "not equal");
// Prints: EQUAL

?>
```

If you're comparing passwords (or anything else for which "near" precision isn't good enough) this confusion could be detrimental. Stick with strict comparisons...

```
<?php
```

```
// Same examples as above, using === instead of ==

echo ("007" === "7" ? "EQUAL" : "not equal");
// Prints: not equal

echo ( (string)'0001' === (string)'+1.' ? "EQUAL" : "not equal");
// Prints: not equal

echo ('  131e-2' === '001.3100' ? "EQUAL" : "not equal");
// Prints: not equal

?>
```

up
down
1

wbcarts at juno dot com ¶

2 years ago

COMPARING PHP OBJECTS (compound type)

We have seen that PHP does a lot of type-juggling on its own -- which can wreak havoc in unexpected ways -- but it is still up to us to produce code that is clear, maintainable AND follows the rules we want to follow.

When creating a PHP Object, it is sometimes unclear what makes two of them the same. But the good part is that we can say what is equal and what is not equal. For example, let's say we have a Student class that includes an equals() method which defines what is equal for this type of object.

```
<?php

#Student.php

class Student
{
  /*
   * These variables are protected to prevent outside code
   * from tampering with them.
   */
  protected $student_id;
  protected $student_name;

  public function __construct($id, $name)
  {
```

```
    $this->student_id = (int)$id;          // cast to integer here
    $this->student_name = (string)$name;   // cast to string here
  }

  /*
   * This function requires an instance of type Student and
   * only evaluates two integers that we set in __construct().
   */
  public function equals(Student $student)
  {
    return ($this->getId() == $student->getId());
  }

  public function getId()
  {
    return $this->student_id;
  }

  public function getName()
  {
    return $this->student_name;
  }

  public function __toString()
  {
    return 'Student [id=' . $this->getId() .
    ', name=' . $this->getName() . ']';
  }
}
?>
```

With this class, the protected variables cannot be tampered with by outside code. Also, the __construct() function casts the variables to the PHP primitives WE WANT, while the equals(Student $student) function, requires an argument of type Student -- which eliminates the need for an IDENTITY '===' check AND prevents any other data types from coming in. One other note: notice how the equals() function only evaluates the $student_id, this allows for two students to have the same name -- which is totally possible.

Here's a short example -- we'll do it correctly AND try to screw it up!

```
<?php

require('Student.php');
```

```
$s1 = new Student(122, 'John Doe');
$s2 = new Student(344, 'John Doe');

echo $s1 . '<br>';  // Student [id=122, name=John Doe]
echo $s2 . '<br>';  // Student [id=344, name=John Doe]

# Check for equality the CORRECT way...
echo ($s1->equals($s2) ? 'EQUAL' : 'NOT EQUAL');  // NOT EQUAL

# Check for equality by HACKING the known value of $student_id...
echo ($s1->equals(122) ? 'EQUAL' : 'NOT EQUAL');  // Catchable fatal error: Argument 1
passed to Student::equals() must be an instance of Student, integer given... etc, etc.

?>
```

See what I mean by writing code that follows OUR RULES? The Student class does the kind of type-juggling we want (and when we want it done) -- NOT when, where, or why PHP does it (not that there's anything wrong with it).

up
down
2

Harry Willis ¶

7 months ago

I was interested about the following two uses of the ternary operator (PHP >= 5.3) for using a "default" value if a variable is not set or evaluates to false:

```
<?php
(isset($some_variable) && $some_variable) ? $some_variable : 'default_value';

$some_variable ?: 'default_value';
?>
```

The second is more readable, but will throw an ERR_NOTICE is $some_variable is not set. Of course, this could be overcome by suppressing the notice using the @ operator.

Performance-wise, though, comparing 1 million iterations of the three statements

```
  (isset($foo) && $foo) ? $foo : ''
  ($foo) ?: ''
  (@$foo) ?: ''
```

results in the following:

```
  $foo is NOT SET.
    [isset] 0.18222403526306
    [?:]    0.57496404647827
    [@ ?:]  0.64780592918396
  $foo is NULL.
    [isset] 0.17995285987854
    [?:]    0.15304207801819
    [@ ?:]  0.20394206047058
  $foo is FALSE.
    [isset] 0.19388508796692
    [?:]    0.15359902381897
    [@ ?:]  0.20741701126099
  $foo is TRUE.
    [isset] 0.17265486717224
    [?:]    0.11773896217346
    [@ ?:]  0.16193103790283
```

In other words, using the long-form ternary operator with isset($some_variable) is preferable overall if $some_variable may not be set.

(error_reporting was set to zero for the benchmark, to avoid printing a million notices...)

up
down
1

gondo ¶

6 months ago

beware of the fact, that there is no `<==` nor `>==` therefore `false <= 0` will be `true`. php v. 5.4.27

up
down
1

sgurukrupa at gmail dot com ¶

7 months ago

With respect to using the ternary operator as a 'null-coalescing' operator: expr1 ?: expr2, note that expr1 is evaluated only once.

up
down
0

mail at mkharitonov dot net ¶

8 months ago

Be careful with the "==" operator when both operands are strings:

```
<?php
var_dump('123' == '       123'); // true
```

```
var_dump('1e3' == '1000'); // true
var_dump('+74951112233' == '74951112233'); // true
var_dump('00000020' == '0000000000000000020'); // true
var_dump('0X1D' == '29E0'); // true
var_dump('0xafebac' == '11529132'); // true
var_dump('0xafebac' == '0XAFEBAC'); // true
var_dump('0xeb' == '+235e-0'); // true
var_dump('0.235' == '+.235'); // true
var_dump('0.2e-10' == '2.0E-11'); // true
var_dump('615295194528097206937025831268l4' == '6152951945280972000000000000000000'); //
true in php < 5.4.4
```

up
down
0

Jeremy Swinborne ¶

2 years ago

Beware of the consequences of comparing strings to numbers. You can disprove the laws of the universe.

```
echo ('X' == 0 && 'X' == true && 0 == false) ? 'true == false' : 'sanity prevails';
```

This will output 'true == false'. This stems from the use of the UNIX function strtod() to convert strings to numbers before comparing. Since 'X' or any other string without a number in it converts to 0 when compared to a number, 0 == 0 && 'X' == true && 0 == false

up
down
0

email at kleijn dot jp ¶

3 years ago

Maybe i am overlooking something but it seems to me that using unset(string) inside a ternary operator creates an error.

```
(($var1==0 && $var2==0)?unset($var3):$var3=$var1+$var2);

result:
Parse error: syntax error, unexpected T_UNSET

using the traditional form of IF...ELSE works normal.

if($var1==0 && $var2==0) { unset($var3); }
else { $var3=$var1+$var2; }

result:
This unsets var3 or creates a sum of var1+var2 for var3
```

JP Kleijn
Netherlands
up
down
-1
pinkgothic at gmail dot com ¶
5 years ago
"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)."

The example covers this behaviour, but it isn't immediately obvious, so:

If you're doing loose comparisons in PHP, note that they differ from checking each value individually like $value1==$value2 by adding what amounts to an empty($value1)==empty($value2) check into the mix. I found this out by investigating some (to me) bizarre behaviour.
[Note that the example contains no ==, just > and <. It's its absence that perceivedly 'causes empty() to fire'.]

I was also pleasantly surprised to see PHP recurse. Also clear if you keep in mind that the example implies another function call to itself with > and < if both operands are arrays, but IMO definitely worth stating.

It might also be worth noting that the order of array keys doesn't matter, even if a foreach() would see a 'different' array. Again, covered by the example, but might be worth stressing.
up
down
-1
Anonymous ¶
5 years ago
Here is some ternary trick I like to use for selecting a default value in a set of radio buttons. This example assumes that a prior value was known and that we are offering a user the chance to edit that prior value. If no prior value was actually known, no default value will be set.

```
<form>
<input type='radio' name='gender' value='m' <?=($gender=='m')?"checked":""?>>Male
<input type='radio' name='gender' value='f' <?=($gender=='f')?"checked":""?>>Female
</form>
```

When a "=" directly follows a "<?" (no space allowed in between -- the trick does not work with "<?php"), the right side of the operand (here, the result of the ternary operation)

is printed out as text into the surrounding HTML code. If using "<?php" form, you will need to do "<?php echo exp1?exp2:exp3 ?>" instead.

up
down
-2

me at lx dot sg ¶

4 years ago

Replying to the comment on Aug 6, 2010, the comparisons return TRUE because they are recognized as numerical strings and are converted to integers. If you try "abc" == " abc", it will return FALSE as expected. To avoid the type conversions, simply use the identity operator (===).

up
down
-1

Boolean_Type ¶

7 months ago

Nice and helpful article!)
I would like to ask:

number == null - it converts both types of comparisons to a boolean or numeric type? In the table the author pointed out that to a boolean. But elsewhere I read that to a numeric type.

up
down
-1

ISAWHIM ¶

4 years ago

When it comes to formatting structure of the conditional statements, I found this to work best and retain logic in views...

```
<?php
$z = 2;
$text = ($z===1 ? 'ONE'
  : ($z===2 ? 'TWO'
  : ($z===3 ? 'THREE'
  : 'MORE' )));
echo($text); // RESULT='TWO'

// LONGHAND

$z = 2;
$text = ($z===1?'ONE' : ($z===2?'TWO' : ($z===3?'THREE' : 'MORE')));
echo($text); // RESULT='TWO'
?>
```

```
Since this is expected to test logic, and nothing more, only use it to test logic.

To test order, if you ever forget...

<?php
$z = 1;
$text = ($z===1 ? 'FIRST : OUTTER'
  : ($z===1 ? 'SECOND : INNER'
  : ($z===1 ? 'THIRD : LAST'
  : 'FAIL EVAL DEFAULT' )));
echo($text); // RETURN='FIRST : OUTTER'

$z = 2;
$text = ($z===1 ? 'FIRST : OUTTER'
  : ($z===1 ? 'SECOND : INNER'
  : ($z===1 ? 'THIRD : LAST'
  : 'FAIL EVAL DEFAULT' )));
echo($text); // RETURN='FAIL EVAL DEFAULT'
?>

(IF ? THEN : ELSE)
(IF ? THEN : ELSE(IF ? THEN : ELSE(IF ? THEN : ELSE))

That can't be read from inside to out, unlike a math formula, because the logic in
comparison is not the same. In math nesting, you need the solution to the deepest nested
element first. In logic comparison, you always start outside before you compare inside.
(Logically, IF there is no door THEN you need something ELSE to get inside. Oh, there is a
window... We are inside, now IF there is a fridge THEN open it or ELSE you starve.)
```

up
down
-1

Amarog ¶

6 years ago

```
Most of the time, you may be content with your conditionals evaluating to true if they are
evaluating a non-false, non-zero value. You may also like it when they evaluate to false
when you use the number 0.

However, there may be times where you want to make a distinction between a non-false value
and a boolean true. You may also wish to make a distinction between a boolean false and a
zero.

The identity operator can make this distinction for you.
```

```
<?php
$a = 'some string';
$b = 123;
$c = 0;

if($a && $b && (!$c))
{ echo "True.\n"; } else { echo "False.\n"; }

if($a == true && $b == true && $c == false)
{ echo "True.\n"; } else { echo "False.\n"; }

if($a === true || $b === true || $c === false)
{ echo "True.\n"; } else { echo "False.\n"; }
?>
```

The above code outputs the following:
True.
True.
False.

As you can see, in the first two cases, $a and $b are considered true, while $c is considered false. If this wasn't the case, neither of the first two conditionals would have echoed "True."

In the last case, I've cleverly used the || operator to demonstrate that both $a and $b do not evaluate to true with the identity operator, nor does $c evaluate to false.

The === operator can be used to distinguish boolean from non-boolean values.

up
down
-1

pcdinh at phpvietnam dot net ¶

8 years ago

You should be very careful when using == with an result set returned from a query that can be an empty array, multi-dimensional array or a boolean false value (if the query failed to execute). In PHP, an empty array is equivalent to true.

```
<?php
$myArray = array();

// check if there is any error with the query
if ($myArray == false)
{
    echo "Yes";
```

```
}
?>

return Yes

Use === instead.
```

up
down
-3

Mark Simon ¶

3 years ago

```
The use of 5.3's shortened ternary operator allows PHP to coalesce a null or empty value
to an alternative:

$value = $planA ?: $planB;

My own server doesn't yet run 5.3. A nice alternative is to use the "or" operator:

$value = $planA or $value = planB;
```

up
down
-2

monkuar at gmail dot com ¶

5 years ago

```
U can even add a variable on that if u wish:

($Profile['skinstyle']=='0')? $lol = "selected":"";

then call it out.. alot faster. if u use EOF.. and such like on ibp :(
```

up
down
-3

webmaster__AT__digitalanime__DOT__nl ¶

10 years ago

```
WARNING!!!!

Let's say, we have this little script:

<?php
$username = 'Me';
$guestusername = 'Guest';

echo 'Hello, ' . isset($i) ? 'my friend: ' . $username . ', how are you doing?' : 'my
guest, ' . $guestusername . ', please register';
```

```
?>

What you want:
If $i is set, display:
Hello, my friend: Me, how are you doing?
If not, display:
Hello, my guest, Guest, please register

BUT, you DON'T get that result!

If $i is set, you get this:
my friend: Me, how are you doing? (so, there's not "Hello, " before it)
If $i is NOT set, you get this:
my friend: Me, how are you doing?

So... That's the same!

You can solve this by using the "(" and ")" to give priority to the ternary operator:

<?php
$username = 'Me';
$guestusername = 'Guest';

echo 'Hello, ' . (isset($i) ? 'my friend: ' . $username . ', how are you doing?' : 'my
guest, ' . $guestusername . ', please register');
?>

When $i is set, you get this:
Hello, my friend: Me, how are you doing? (expected)
When $i is NOT set, you get this:
Hello, my guest, Guest, please register (expected too)

So.. Please, don't be dumb and ALWAYS use the priority-signs (or.. How do you call them?),
( and ).
By using them, you won't get unneeded trouble and always know for sure your code is doing
what you want: The right thing.
```

up
down
-5

ken at smallboxsoftware net ¶

7 years ago

```
This seems a bit odd to me, but PHP can convert two strings into integers during
comparison.  For example: "700" == "+700" return true even though they are totally
different strings.
```

Use === or strcmp when comparing two strings to ensure that they remain as strings during comparison.

up
down
-5

sven dot heyll at web dot de ¶
9 years ago

Hi folks,
to the float comparison problem...

This worked for me:

```
<?php
//! compare two floating point values, return true if they are equal
//! (enough) or false otherwise
function float_equal($f1, $f2)
{
    return ($f1 > $f2) ? (false) : (!($f1 < $f2));
}

// compare floats
$f1 = 0.037;
$f2 = 1000387.978;
echo "$f1 and $f2 are ".(float_equal($f1,$f2)?("equal"):("not equal"))."<br />";
$f1 = 0.3;
$f2 = 0.3;
echo "$f1 and $f2 are ".(float_equal($f1,$f2)?("equal"):("not equal"))."<br />";
?>
```

up
down
-3

Anonymous ¶
7 years ago

Since php 5.2 the operator == for object vs object is not recursion safe, it will cause a fatal error if one of the objects contains a refernce to it self (indirect refferences also counts here).
If you are just checking if two object pointers points to the same object use === instead and aviod this issue (you might get a minimal speed bost too).

up
down
-5

Hayley Watson ¶
6 years ago

The cast from null to boolean is documented (on the page describing the boolean type: null->false), and so is the cast from boolean to integer (on the page describing the integer type: false->0), but the cast from null to integer is undefined and the fact that it is currently implemented by casting from null to boolean and then from boolean to integer is explicitly documented (on the page describing the integer type) as something that should not be relied on (so null==0 is true only by accident, but ((int)(bool)null)==0 is true per specification).

Perhaps as well as a "Converting to integer" section on the integer type page there should also be a "Converting from integer" section; similarly for the other types.

add a note