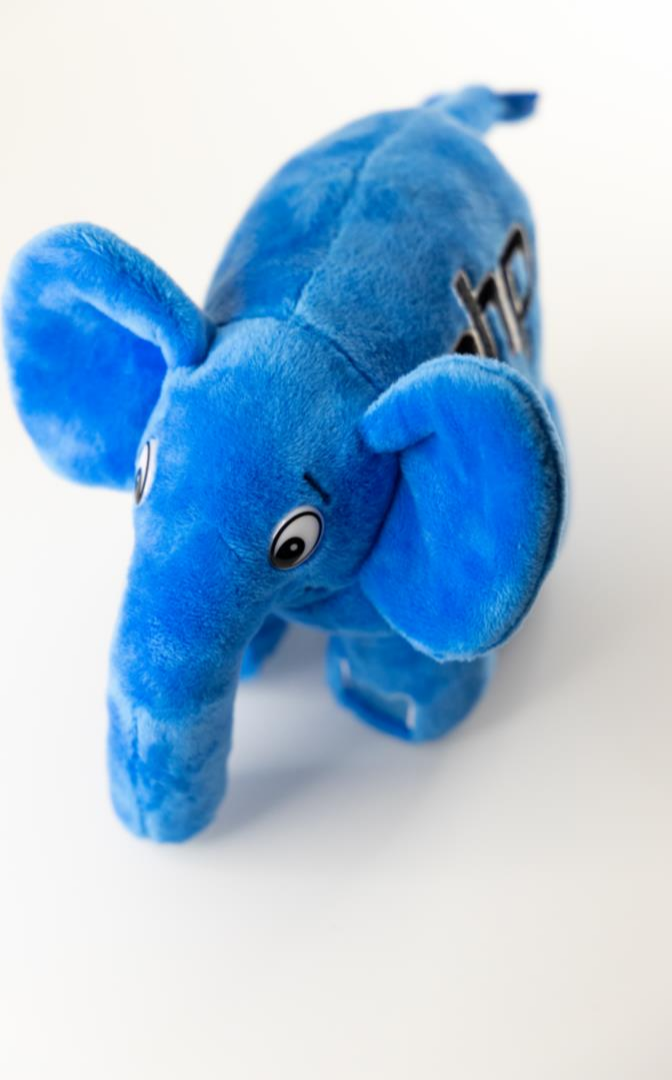
Data Types Detection and Conversion

Disclaimer: вы смотрите просто запись лекции, это **НЕ** специально подготовленный видеокурс!



WRONG way of Data Type detection

Generally, you may use gettype() function:

```php
<?php

// gettype(mixed $value): string
$someVariable = 'Test';
echo gettype($someVariable);

/*
 Possible values for the returned string are:
 "boolean"
 "integer"
 "double" (for historical reasons "double" is returned in case of a float, and not simply "float")
 "string"
 "array"
 "object"
 "resource"
 "resource (closed)" as of PHP 7.2.0
 "NULL"
 "unknown type"
*/
```

Please, do NOT!

Why shouldn’t we ever use **gettype()** function?!

With gettype()

```
<?php

$someVariable = 'Test’;

if (gettype($someVariable) == 'strinng') {
    // This condition is ALWAYS false!
    // Good luck to see why.
}
```

Without gettype()

```
<?php

$someVariable = 'Test';

if (is_strinng($someVariable)) {
    // PHP immediately detects the problem.
}
```

Good way of Data Type detection

So, instead, use these functions:

- is_array() - Finds whether a variable is an array.
- is_bool() - Finds out whether a variable is a boolean.
- is_callable() - Verify that a value can be called as a function from the current scope.
- is_float() - Finds whether the type of a variable is float.
- is_int() - Find whether the type of a variable is integer.
- is_null() - Finds whether a variable is null.
- is_numeric() - Finds whether a variable is a number or a numeric string.
- is_object() - Finds whether a variable is an object.
- is_resource() - Finds whether a variable is a resource.
- is_scalar() - Finds whether a variable is a scalar.
- is_string() - Find whether the type of a variable is string.
- function_exists() - Return true if the given function has been defined.
- method_exists() - Checks if the class method exists.

WRONG way of Data Type conversion

Generally, you may use settype() function:

```php
<?php

// settype(mixed &$var, string $type): bool
$someVariable = '50';
$someVariable = settype($someVariable, "integer");

/*
 Possibles values of type are:
 "boolean" or "bool"
 "integer" or "int"
 "float" or "double"
 "string"
 "array"
 "object"
 "null"
*/
```

Please, do NOT!

Why shouldn’t we ever use **settype()** function?!

With settype()

```php
<?php

$someVariable = '50';
$someVariable = settype($someVariable, "innteger");
// Yes, it fails, still it’s hard to see.
```

Without settype()

```php
<?php

$someVariable = '50';
$someVariable = (innt)$someVariable;
// IDE immediately detects the problem.
```

Good way of Data Type conversion

So, instead, use this classic approach:

- (int), (integer) - cast to int.
- (bool), (boolean) - cast to bool.
- (float), (double), (real) - cast to float.
- (string) - cast to string.
- (array) - cast to array.
- (object) - cast to object.

Type Casting and Type Juggling (two cases of type conversion)

Type casting in PHP works much as it does in C: the name of the desired type is written in parentheses before the variable which is to be cast.

The idea here is that **a variable changes its type**.

```php
<?php

$someVariable = '50'; // string
$someVariable = (int)$someVariable; // int
```

Type Casting and Type Juggling (two cases of type conversion)

Type juggling in PHP allows several variables of different types to participate in a single expression. Only copies of that variables are changes, original variables stay intact.

The idea here is that **a variable preserves its type**.

```php
<?php

$someVariableOne = '1';  // $someVariableOne is a string
$someVariableTwo = $someVariableOne * 2; // $someVariableOne is still a string

$someStringOne = '10 oranges';
$someStringTwo = '2.5 lemons';
$someVariable = 5 * $someStringOne; // $someStringOne is still a string
$someVariable = 10 + $someStringTwo; // $someStringTwo is still a string

// Pay attention! If we assign the result to the same variable,
// it will change its type:
$someVariable = 1; // int
$someVariable = $someVariable * 2.5; // float
```

Casting to boolean

false	true
false	true
0	resource
0.0 and -0.0	NaN
‘’ and ‘0’	Anything else (not mentioned in the left column)
array()	
null	

Casting to boolean

And a simple code to remember it better:

```php
<?php

var_dump((bool)"");          // bool(false)
var_dump((bool)"0");         // bool(false)
var_dump((bool)1);           // bool(true)
var_dump((bool)-2);          // bool(true)
var_dump((bool)"Test");      // bool(true)
var_dump((bool)2.3e5);       // bool(true)
var_dump((bool)array(12));   // bool(true)
var_dump((bool)array());     // bool(false)
var_dump((bool)"false");     // bool(true)
```

Casting to integer

From boolean	From float	From string
false → 0 true → 1	Decimal part is omitted.	See further ☺.
From null	**From NaN**	**From Infinity**
null → 0	NaN → 0	Infinity → 0

Conversion from other types is undefined!

Casting to integer

And a simple code to remember it better:

```php
<?php

var_dump((int)false);        // int(0)
var_dump((int)true);         // int(1)
var_dump((int)10.7);         // int(10)
var_dump((int)null);         // int(0)
var_dump((int)"Test");       // int(0)
var_dump((int)"5 Test");     // int(5)
var_dump((int)"2.5 Test");   // int(2)
var_dump((int)sqrt(-1));     // int(0)
var_dump((int)array(12));    // int(1)
var_dump((int)99e9999);      // int(0)
```

Casting to integer: beware of precision problems!

```php
<?php

echo (int)((0.1 + 0.7) * 10); // echoes 7!

/* Floating point numbers have limited precision.
Although it depends on the system, PHP typically uses the IEEE 754 double precision format,
which will give a maximum relative error due to rounding in the order of 1.11e-16.
Non-elementary arithmetic operations may give larger errors, and, of course, error propagation
must be considered when several operations are compounded.

Additionally, rational numbers that are exactly representable as floating point numbers in base 10,
like 0.1 or 0.7, do not have an exact representation as floating point numbers in base 2,
which is used internally, no matter the size of the mantissa.
Hence, they cannot be converted into their internal binary counterparts without a small loss of precision.
This can lead to confusing results: for example, floor((0.1+0.7)*10) will usually return 7 instead
of the expected 8, since the internal representation will be something like 7.9999999999999991118....

So never trust floating number results to the last digit, and do not compare floating
point numbers directly for equality. If higher precision is necessary, the arbitrary
precision math functions and gmp functions are available. */
```

Casting to float

From boolean	From float	From string
false → 0.0 true → 1.0	No changes ☺.	See further ☺.
From null	**From NaN**	**From Infinity**
null → 0.0	NaN → NaN	Infinity → INF

Conversion from other types is undefined!

Casting to float

And a simple code to remember it better:

```php
<?php

var_dump((float)false);        // double(0.0)
var_dump((float)true);         // double(1.0)
var_dump((float)10.7);         // double(10.7)
var_dump((float)null);         // double(0.0)
var_dump((float)"Test");       // double(0.0)
var_dump((float)"5 Test");     // double(5.0)
var_dump((float)"2.5 Test"); // double(2.5)
var_dump((float)sqrt(-1));     // double(NaN)
var_dump((float)array(12));    // double(1.0)
var_dump((float)99e9999);      // double(INF)
```

Numeric strings (and casting to integers or doubles)

We've already seen this in a previous topics…

A PHP string is considered numeric if it can be interpreted as an int or a float.

```
<?php
$x = 1 + "10.5";                  // $x is float (11.5)
$x = 1 + "-1.3e3";                // $x is float (-1299)
$x = 1 + "bob-1.3e3";             // TypeError as of PHP 8.0.0, $x is integer (1) previously
$x = 1 + "bob3";                  // TypeError as of PHP 8.0.0, $x is integer (1) previously

$x = 1 + "10 Small Pigs";         // $x is integer (11) and an E_WARNING is raised in PHP 8.0.0,
                                  // E_NOTICE previously

$x = 4 + "10.2 Little Piggies"; // $x is float (14.2) and an E_WARNING is raised in PHP 8.0.0,
                                  // E_NOTICE previously

$x = "10.0 pigs " + 1;            // $x is float (11) and an E_WARNING is raised in PHP 8.0.0,
                                  // E_NOTICE previously

$x = "10.0 pigs " + 1.0;          // $x is float (11) and an E_WARNING is raised in PHP 8.0.0,
                                  // E_NOTICE previously
```

Casting to string

From boolean

Yes! Empty string! Not “0”!

false → ‘’

true → ‘1’

From array, resource, object

array → ‘Array’ + ***WARNING***

object → ***ERROR***

resource → ‘Resource id # N’ + ***WARNING***

From null

null → ‘’

Implicit casting to string happens each time you output a variable to some text stream (console, web-server, etc…)

Casting to string

And a simple code to remember it better:

```php
<?php

var_dump((string)false);          // string(0) ""
var_dump((string)true);           // string(1) "1"
var_dump((string)10.7);           // string(4) "10.7"
var_dump((string)null);           // string(0) ""
var_dump((string)"Test");         // string(4) "Test"
var_dump((string)"5 Test");       // string(6) "5 Test"
var_dump((string)"2.5 Test");     // string(8) "2.5 Test"
var_dump((string)sqrt(-1));       // string(3) "NAN"
var_dump((string)array(12));      // string(5) "Array" + WARNING
var_dump((string)99e9999);        // string(3) "INF"

$fileResource = fopen($_SERVER['PHP_SELF'], 'rb');
var_dump((string)$fileResource); // "Resource id #5" + WARNING

class SomeClass{};
var_dump((string)(new SomeClass)); // ERROR
```

Casting to array

From simple types and resource

An array with one element holding the initial value

From object

See the documentation, there's a lot of uncertainty…

From null

An empty array

Casting to array

And a simple code to remember it better:

```
<?php

var_dump((array)false);          // array(1) {[0] => bool(false)}
var_dump((array)true);           // array(1) {[0] => bool(true)}
var_dump((array)10.7);           // array(1) {[0] => double(10.7)}
var_dump((array)null);           // array(0) {}
var_dump((array)"Test");         // array(1) {[0] =>  string(4) "Test"}
var_dump((array)"5 Test");       // array(1) {[0] =>  string(6) "5 Test"}
var_dump((array)"2.5 Test");     // array(1) {[0] =>  string(8) "2.5 Test"}
var_dump((array)sqrt(-1));       // array(1) {[0] =>  double(NAN)}
var_dump((array)array(12));      // array(1) {[0] =>  int(12)}
var_dump((array)99e9999);        // array(1) {[0] =>  double(INF)}

$fileResource = fopen($_SERVER['PHP_SELF'], 'rb');
var_dump((array)$fileResource); // array(1) {[0] =>  resource(5) of type (stream)}

class SomeClass{};
var_dump((array)(new SomeClass)); // array(0) {}
```

Casting to object

From any type except null and array

An object with a single "**scalar**" property (holding the initial value).

From null

An empty object (without properties).

From array

An object with properties named by keys and corresponding values.

Casting to object

And a simple code to remember it better:

```
<?php

var_dump((object)false);              // class stdClass#1 (1) {public $scalar =>  bool(false)}
var_dump((object)true);               // class stdClass#1 (1) {public $scalar =>  bool(true)}
var_dump((object)10.7);               // class stdClass#1 (1) {public $scalar =>  double(10.7)}
var_dump((object)null);               // class stdClass#1 (0) {}
var_dump((object)"Test");             // class stdClass#1 (1) {public $scalar =>  string(4) "Test"}
var_dump((object)"5 Test");           // class stdClass#1 (1) {public $scalar =>  string(6) "5 Test"}
var_dump((object)"2.5 Test");         // class stdClass#1 (1) {public $scalar =>  string(8) "2.5 Test"}
var_dump((object)sqrt(-1));           // class stdClass#1 (1) {public $scalar =>  double(NAN)}
var_dump((object)array(12));          // class stdClass#1 (1) {public $0 =>  int(12)}
var_dump((object)array('a' => 'A',
                       'b' => 'B'));  // class stdClass#1 (2) {public $a =>  string(1) "A"
                                      //                       public $b =>  string(1) "B"}
var_dump((object)99e9999);            // class stdClass#1 (1) {public $scalar =>  double(INF)}

$fileResource = fopen($_SERVER['PHP_SELF'], 'rb');
var_dump((object)$fileResource);    // class stdClass#1 (1) {public $scalar =>  resource(5) of type (stream)}

class SomeClass{};
var_dump((object)(new SomeClass)); // class SomeClass#1 (0) {}
```

Casting to resource and null

From any type to resource

ERROR

It literally makes no sense. It's like… converting an orange to milliseconds.

From any type to null

ERROR

Use **unset()** instead!

Several useful expressions with results

Expression	gettype()	empty()	is_null()	isset()	boolean : if($x)
$x = "";	string	true	false	true	false
$x = null	null	true	true	false	false
var $x;	null	true	true	false	false
$x is not defined	null	true	true	false	false
$x = array();	array	true	false	true	false
$x = false;	boolean	true	false	true	false
$x = true;	boolean	false	false	true	true
$x = 1;	integer	false	false	true	true
$x = 42;	integer	false	false	true	true
$x = 0;	integer	true	false	true	false
$x = -1;	integer	false	false	true	true
$x = "1";	string	false	false	true	true
$x = "0";	string	true	false	true	false
$x = "-1";	string	false	false	true	true
$x = "php";	string	false	false	true	true
$x = "true";	string	false	false	true	true
$x = "false";	string	false	false	true	true

Comparison by == results

	true	false	1	0	-1	"1"	"0"	"-1"	null	array()	"php"	""
true	true	false	true	false	true	true	false	true	false	false	true	false
false	false	true	false	true	false	false	true	false	true	true	false	true
1	true	false	true	false	false	true	false	false	false	false	false	false
0	false	true	false	true	false	false	true	false	true	false	true	true
-1	true	false	false	false	true	false	false	true	false	false	false	false
"1"	true	false	true	false	false	true	false	false	false	false	false	false
"0"	false	true	false	true	false	false	true	false	false	false	false	false
"-1"	true	false	false	false	true	false	false	true	false	false	false	false
null	false	true	false	true	false	false	false	false	true	true	false	true
array()	false	true	false	false	false	false	false	false	true	true	false	false
"php"	true	false	false	true	false	false	false	false	false	false	true	false
""	false	true	false	true	false	false	false	false	true	false	false	true

Comparison by === results

	true	false	1	0	-1	"1"	"0"	"-1"	null	array()	"php"	""
true	true	false	false	false	false	false	false	false	false	false	false	false
false	false	true	false	false	false	false	false	false	false	false	false	false
1	false	false	true	false	false	false	false	false	false	false	false	false
0	false	false	false	true	false	false	false	false	false	false	false	false
-1	false	false	false	false	true	false	false	false	false	false	false	false
"1"	false	false	false	false	false	true	false	false	false	false	false	false
"0"	false	false	false	false	false	false	true	false	false	false	false	false
"-1"	false	false	false	false	false	false	false	true	false	false	false	false
null	false	false	false	false	false	false	false	false	true	false	false	false
array()	false	false	false	false	false	false	false	false	false	true	false	false
"php"	false	false	false	false	false	false	false	false	false	false	true	false
""	false	false	false	false	false	false	false	false	false	false	false	true

Check yourself! What is the $c value in each case?

```php
<?php

// Case 1:
$a = "10 cats";
$b = "5$a dogs";
$c = $a / $b;
var_dump($c);

// Case 2:
$a = "10 cats";
$b = '5$a dogs';
$c = $a / $b;
var_dump($c);

// Case 3:
$a = true;
$b = "5$a dogs";
$c = $a / $b;
var_dump($c);
```

What cases would produce warning messages?

Check yourself! What is the $c value in each case?

```php
<?php

// Case 1:
$a = "10 cats";
$b = "5$a dogs";
$c = $a / $b;
var_dump($c); // double(0.019607843137255) (1/51) + WARNNIG

// Case 2:
$a = "10 cats";
$b = '5$a dogs';
$c = $a / $b;
var_dump($c); // int(2) + WARNING

// Case 3:
$a = true;
$b = "5$a dogs";
$c = $a / $b;
var_dump($c); // double(0.019607843137255) (1/51) + WARNING
```

Data Types Detection and Conversion

Disclaimer: вы смотрите просто запись лекции, это **НЕ** специально подготовленный видеокурс!