Type Operators

`instanceof` is used to determine whether a PHP variable is an instantiated object of a certain class:

Using `instanceof` with classes

```
<?php
class MyClass
{
}

class NotMyClass
{
}
$a = new MyClass;

var_dump($a instanceof MyClass);
var_dump($a instanceof NotMyClass);
?>
```

ENTITY

```
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:

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);
?>
```

ENTITY

```
bool(true)
bool(true)
```

To check if an object is *not* an instanceof a class, the logical `not` operator can be used.

Using `instanceof` to check if object is *not* an instanceof a class

```
<?php
class MyClass
{
}

$a = new MyClass;
var_dump(!($a instanceof stdClass));
?>
```

ENTITY

```
bool(true)
```

Lastly, `instanceof` can also be used to determine whether a variable is an instantiated object of a class that implements an interface:

Using `instanceof` with interfaces

```
<?php
interface MyInterface
{
}

class MyClass implements MyInterface
{
}

$a = new MyClass;

var_dump($a instanceof MyClass);
var_dump($a instanceof MyInterface);
?>
```

ENTITY

```
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:

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'
?>
```

ENTITY

```
bool(true)
bool(true)
bool(false)
```

instanceof does not throw any error if the variable being tested is not an object, it simply returns ENTITY. Constants, however, were not allowed prior to PHP 7.3.0.

Using `instanceof` to test other variables

```
<?php
$a = 1;
$b = NULL;
$c = fopen('/tmp/', 'r');
var_dump($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);
?>
```

ENTITY

```
bool(false)
bool(false)
bool(false)
PHP Fatal error:  instanceof expects an object instance, constant given
```

As of PHP 7.3.0, constants are allowed on the left-hand-side of the `instanceof` operator.

Using `instanceof` to test constants

```
<?php
var_dump(FALSE instanceof stdClass);
?>
```

ENTITY

```
bool(false)
```

As of PHP 8.0.0, `instanceof` can now be used with arbitrary expressions. The expression must be wrapped in parentheses and produce a `string`.

Using `instanceof` with an arbitrary expression

```
<?php

class ClassA extends \stdClass {}
class ClassB extends \stdClass {}
class ClassC extends ClassB {}
class ClassD extends ClassA {}

function getSomeClass(): string
{
    return ClassA::class;
}

var_dump(new ClassA instanceof ('std' . 'Class'));
var_dump(new ClassB instanceof ('Class' . 'B'));
var_dump(new ClassC instanceof ('Class' . 'A'));
var_dump(new ClassD instanceof (getSomeClass()));
?>
```

ENTITY

```
bool(true)
bool(true)
bool(false)
bool(true)
```

The `instanceof` operator has a functional variant with the `is_a` function.

ENTITY

`get_class`, `is_a`

□□□ #2
□ □□□ □□□ 2026-10-31 17:41:31 UTC
□ □□□ □□□ 2026-10-31 17:41:32 UTC