




- [Downloads](#)
- [Documentation](#)
- [Get Involved](#)
- [Help](#)
- 

[Dutch PHP Conference 2024](#)

## [Getting Started](#)

[Introduction](#)

[A simple tutorial](#)

## [Language Reference](#)

[Basic syntax](#)

[Types](#)

[Variables](#)

[Constants](#)

[Expressions](#)

[Operators](#)

[Control Structures](#)

[Functions](#)

[Classes and Objects](#)

[Namespaces](#)

[Enumerations](#)

[Errors](#)

[Exceptions](#)

[Fibers](#)

[Generators](#)

[Attributes](#)

[References Explained](#)

[Predefined Variables](#)

[Predefined Exceptions](#)

[Predefined Interfaces and Classes](#)

[Predefined Attributes](#)

[Context options and parameters](#)

[Supported Protocols and Wrappers](#)

## [Security](#)

[Introduction](#)

[General considerations](#)

[Installed as CGI binary](#)

[Installed as an Apache module](#)

[Session Security](#)

[Filesystem Security](#)

[Database Security](#)

[Error Reporting](#)

[User Submitted Data](#)

[Hiding PHP](#)

[Keeping Current](#)

## [Features](#)

[HTTP authentication with PHP](#)

[Cookies](#)

[Sessions](#)

[Dealing with XForms](#)

[Handling file uploads](#)

[Using remote files](#)

[Connection handling](#)

[Persistent Database Connections](#)

[Command line usage](#)

[Garbage Collection](#)

[DTrace Dynamic Tracing](#)

## [Function Reference](#)

[Affecting PHP's Behaviour](#)

[Audio Formats Manipulation](#)

[Authentication Services](#)

[Command Line Specific Extensions](#)

[Compression and Archive Extensions](#)

- [Cryptography Extensions](#)
- [Database Extensions](#)
- [Date and Time Related Extensions](#)
- [File System Related Extensions](#)
- [Human Language and Character Encoding Support](#)
- [Image Processing and Generation](#)
- [Mail Related Extensions](#)
- [Mathematical Extensions](#)
- [Non-Text MIME Output](#)
- [Process Control Extensions](#)
- [Other Basic Extensions](#)
- [Other Services](#)
- [Search Engine Extensions](#)
- [Server Specific Extensions](#)
- [Session Extensions](#)
- [Text Processing](#)
- [Variable and Type Related Extensions](#)
- [Web Services](#)
- [Windows Only Extensions](#)
- [XML Manipulation](#)
- [GUI Extensions](#)

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

[Манипуляции с типами »](#)  
[« Итерируемые значения](#)

- [Руководство по PHP](#)
- [Справочник языка](#)
- [Типы](#)

Change language: 

Russian

[Submit a Pull Request](#) [Report a Bug](#)

Объявления типов

Объявления типов разрешено добавлять к аргументам функции, возвращаемым значениям и свойствам класса (последнее начиная с PHP 7.4.0). Объявленные типы гарантируют, что во время вызова значение принадлежит тому типу, который для него указан, иначе будет выброшено исключение [TypeError](#).

Каждый тип, который поддерживает PHP, за исключением ресурсов (resource), разрешено указывать при пользовательском объявлении типа. На этой странице приведён журнал изменений доступности отдельных типов и документация о том, как их применять в объявлениях типов.

Замечание:

Когда класс реализует метод интерфейса или переопределяет метод, который уже был определён родительским классом, вновь определяемый метод должен быть совместим с определением, которое было сделано ранее. Метод совместим, если он соблюдает [правила вариантности](#).

## Список изменений

Версия	Описание
8.3.0	Добавлена поддержка типизации констант классов, интерфейсов, трейтов и перечислений.
8.2.0	Добавлена поддержка типов DNF.
8.2.0	Добавлена поддержка типа true.
8.2.0	Типы null и false теперь можно использовать автономно.
8.1.0	Добавлена поддержка пересечений типов.
8.1.0	Возврат по ссылке из функции с типом возвращаемого значения void устарел.
8.1.0	Добавлена поддержка типа возвращаемого значения never.
8.0.0	Добавлена поддержка типа возвращаемого значения <a href="#">mixed</a> .
8.0.0	Добавлена поддержка типа возвращаемого значения static.
8.0.0	Добавлена поддержка объединения типов.
7.4.0	Добавлена поддержка типизации свойств классов.
7.2.0	Добавлена поддержка типа возвращаемого значения object.
7.1.0	Добавлена поддержка типа возвращаемого значения <a href="#">iterable</a> .
7.1.0	Добавлена поддержка типа возвращаемого значения void.
7.1.0	Добавлена поддержка типа возвращаемого значения nullable.

## Примечание по использованию атомарных типов

У атомарных типов прямолинейное поведение с незначительными оговорками, которые описаны в этом разделе.

### Скалярные типы

#### Внимание

Псевдонимы имён для скалярных типов (bool, int, float, string) не поддерживаются. Вместо этого они рассматриваются как имена классов или интерфейсов. Например, когда в качестве типа указан boolean, ожидается, что значение выполняет условие [instanceof](#) в отношении класса или интерфейса boolean, а не значение типа bool:

```
<?php
function test(boolean $param) {}
test(true);
?>
```

Результат выполнения приведённого примера в PHP 8:

```
Warning: "boolean" will be interpreted as a class name. Did you mean "bool"? Write "\boolean" to suppress this warning in /in/9YrUX on line 2

Fatal error: Uncaught TypeError: test(): Argument #1 ($param) must be of type boolean, bool given, called in - on line 3 and defined in -:2
Stack trace:
#0 -(3): test(true)
#1 {main}
   thrown in - on line 2
```

### void

#### Замечание:

Возврат по ссылке из void-функции устарел начиная с PHP 8.1.0, поскольку такая функция противоречива. Ранее при её вызове выдавалась ошибка уровня `E_NOTICE`: Только ссылки на переменные должны возвращаться по ссылке.

```
<?php
function &test(): void {}
?>
```

### Тип Callable

Этот тип нельзя объявлять в качестве типа свойства класса.

**Замечание:** Этот тип невозможно указать в качестве названия функции.

### Объявление типов в параметрах передачи по ссылкам

Если у параметра, передаваемого по ссылке, объявляется тип возвращаемого значения, тип переменной проверяется *только* при входе в функцию, в начале вызова, но не при возврате функции. То есть функция может изменить тип ссылки на переменную.

#### Пример #1 Типизированные параметры, передаваемые по ссылке

```
<?php
```

```
function array_baz(array &$param)
{
    $param = 1;
}
$var = [];
array_baz($var);
var_dump($var);
array_baz($var);
?>
```

Вывод приведённого примера будет похож на:

```
int(1)

Fatal error: Uncaught TypeError: array_baz(): Argument #1 ($param) must be of type array, int given, called in - on line 9 and defined in -:2
Stack trace:
#0 -(9): array_baz(1)
#1 {main}
    thrown in - on line 2
```

Примечание по составным типам

На объявления составных типов распространяется ряд ограничений и во время компиляции PHP выполнит проверку объявленных типов на избыточность, чтобы предотвратить простые ошибки.

Предостережение

До PHP 8.2.0 и появления DNF-типов, было невозможно комбинировать пересечённые и объединённые типы.

Объединение типов

Внимание

Невозможно объединять два типа значений false и true. Вместо этого указывают bool.

Предостережение

Поскольку до PHP 8.2.0 нельзя было определять false и null как отдельные типы, объединение типов, которое состояло только из этих типов, было недопустимо. Сюда входят типы: false, false|null и ?false.

Синтаксический сахар типа Nullable

Объявление одного базового типа может быть помечено как nullable путём добавления к типу префикса в виде вопросительного знака (?). Поэтому ?T и T|null идентичны.

**Замечание:** Этот синтаксис поддерживается с PHP 7.1.0 и предшествует поддержке объединения типов.

Замечание:

Ещё одним способом добиться nullable-аргументов — указать null значением по умолчанию. Такой способ не рекомендован, поскольку если значение по умолчанию будет изменено в дочернем классе, возникнет нарушение совместимости типов, так как в объявление типа нужно будет добавить тип null.

Пример #2 Старый способ указания nullable-аргументов

```
<?php

class C {}

function f(C $c = null) {
    var_dump($c);
}

f(new C);
f(null);
?>
```

Результат выполнения приведённого примера:

```
object(C)#1 (0) {
}
NULL
```

Повторяющиеся и избыточные типы

Избыточные типы, которые можно обнаружить без выполнения загрузки класса, приведут к ошибке во время компиляции, чтобы отловить неточности в объявлениях составных типов. В них включены:

- Каждое имя после разрешения внутренними средствами языка, может встречаться только один раз. Например, типы `int|string|INT` или `Countable&Traversable&COUNTABLE` приведут к ошибке.
- Указание типа [mixed](#) (с другими типами) приведёт к ошибке.
- Для объединённых типов:
  - Если указан тип `bool`, то `false` или `true` не могут быть указаны дополнительно.
  - Если указан тип `object`, типы классов нельзя указывать дополнительно.
  - Если указан тип [iterable](#), то `array` и [Traversable](#) нельзя указывать дополнительно.
- Для пересечённых типов:
  - Указание типа, который не относится к типу класса, приведёт к ошибке.
  - Указание `self`, `parent` или `static` приведёт к ошибке.
- Для DNF-типов:
  - Если указан более общий тип, то более строгий тип будет избыточным.
  - Дублирование членов в пересечённых типах.

**Замечание:** Это не гарантирует, что тип «минимальный», поскольку для этого пришлось бы загрузить все указанные типы классов.

Например, если `A` и `B` — это псевдонимы классов, то `A|B` остаётся корректным объединением типов, даже если его можно свести либо к `A`, либо к `B`. Аналогично, если класс `B extends A {}`, то `A|B` также является корректным объединением типов, даже если его можно свести к просто `A`.

```
<?php

function foo(): int|INT {} // Запрещено
function foo(): bool>false {} // Запрещено
function foo(): int&Traversable {} // Запрещено
function foo(): self&Traversable {} // Запрещено

use A as B;
function foo(): A|B {} // Запрещено («use» — часть разрешения имён)
function foo(): A&B {} // Запрещено («use» — часть разрешения имён)

class_alias('X', 'Y');
function foo(): X|Y {} // Разрешено (избыточность известна только во время выполнения)
function foo(): X&Y {} // Разрешено (избыточность известна только во время выполнения)
?>
```

## Примеры

### Пример #3 Пример объявления типа класса

```
<?php

class C {}
class D extends C {}

// Не наследует C.
class E {}

function f(C $c) {
    echo get_class($c)."\n";
}

f(new C);
f(new D);
f(new E);
?>
```

Результат выполнения приведённого примера в PHP 8:

```
C
D

Fatal error: Uncaught TypeError: f(): Argument #1 ($c) must be of type C, E given, called in /in/gLonb on line 14 and defined in /in/gLonb:8
Stack trace:
#0 -(14): f(Object(E))
#1 {main}
  thrown in - on line 8
```

### Пример #4 Пример объявления типа интерфейса

```
<?php

interface I { public function f(); }
class C implements I { public function f() {} }
```

```
// Не реализует I.
class E {}

function f(I $i) {
    echo get_class($i)."\n";
}

f(new C);
f(new E);
?>
```

Результат выполнения приведённого примера в PHP 8:

```
C
```

```
Fatal error: Uncaught TypeError: f(): Argument #1 ($i) must be of type I, E given, called in - on line 13 and defined in -:8
Stack trace:
#0 - (13): f(Object(E))
#1 {main}
   thrown in - on line 8
```

#### Пример #5 Пример объявления типа возвращаемого значения

```
<?php

function sum($a, $b): float {
    return $a + $b;
}

// Обратите внимание, что будет возвращено значение float.
var_dump(sum(1, 2));
?>
```

Результат выполнения приведённого примера:

```
float(3)
```

#### Пример #6 Возвращение объекта

```
<?php

class C {}

function getC(): C {
    return new C;
}

var_dump(getC());
?>
```

Результат выполнения приведённого примера:

```
object(C)#1 (0) {
}
```

#### Пример #7 Объявление аргумента с типом Nullable

```
<?php

class C {}

function f(?C $c) {
    var_dump($c);
}

f(new C);
f(null);
?>
```

Результат выполнения приведённого примера:

```
object(C)#1 (0) {
}
NULL
```

#### Пример #8 Объявление типа возвращаемого значения Nullable

```
<?php

function get_item(): ?string {
if (isset($_GET['item'])) {
return $_GET['item'];
} else {
return null;
}
}
?>
```

#### Пример #9 Объявление типа свойства класса

```
<?php

class User {
public static string $foo = 'foo';

public int $id;
public string $username;

public function __construct(int $id, string $username) {
$this->id = $id;
$this->username = $username;
}
}
?>
```

### Строгая типизация

По умолчанию PHP будет преобразовывать значения неправильного типа в ожидаемые. Например, если в строковый (string) параметр функции передать целое число (int), то оно преобразуется в строку (string).

Можно включить режим строгой типизации на уровне файла. В этом режиме, тип значения должен строго соответствовать объявленному, иначе будет выброшено исключение [TypeError](#). Единственное исключение из этого правила — передача целочисленного значения (int) туда, где ожидается число с плавающей точкой (float).

#### Внимание

На вызовы из внутренних функций действие strict\_types не распространяется.

Для включения строгой типизации указывают оператор [declare](#) с объявлением strict\_types:

#### Замечание:

Строгая типизация распространяется на вызовы функций, которые сделаны *изнутри* файла с включённой строгой типизацией, а не к функциям, объявленным в этом файле. Если из файла без включённой строгой типизации вызывается функция, которая была определена в файле со строгой типизацией, то будут использованы установки по типизации вызывающей стороны — т. е. правила строгой типизации будут проигнорированы и для значений будет применяться приведение типов.

#### Замечание:

Строгая типизация определяется только для объявлений скалярных типов.

#### Пример #10 Строгая типизация для значений аргументов

```
<?php

declare(strict_types=1);

function sum(int $a, int $b) {
return $a + $b;
}

var_dump(sum(1, 2));
var_dump(sum(1.5, 2.5));
?>
```

Результат выполнения приведённого примера в PHP 8:

```
int(3)

Fatal error: Uncaught TypeError: sum(): Argument #1 ($a) must be of type int, float given, called in - on line 9 and defined in -:4
Stack trace:
#0 -(9): sum(1.5, 2.5)
```

```
#1 {main}
  thrown in - on line 4
```

## Пример #11 Приведение типов для значений аргументов

```
<?php

function sum(int $a, int $b) {
    return $a + $b;
}

var_dump(sum(1, 2));

// Переданные значения будут приведены к целым числам: обратите внимание на вывод ниже!
var_dump(sum(1.5, 2.5));
?>
```

Результат выполнения приведённого примера:

```
int(3)
int(3)
```

## Пример #12 Строгая типизация для возвращаемых значений

```
<?php

declare(strict_types=1);

function sum($a, $b): int {
    return $a + $b;
}

var_dump(sum(1, 2));
var_dump(sum(1, 2.5));
?>
```

Результат выполнения приведённого примера:

```
int(3)

Fatal error: Uncaught TypeError: sum(): Return value must be of type int, float returned in -:5
Stack trace:
#0 -(9): sum(1, 2.5)
#1 {main}
  thrown in - on line 5
```

[+add a note](#)

## User Contributed Notes 2 notes

[up](#)  
[down](#)

19  
[toinenkayt \(ta at ta\) \[iwonderr\] gmail d ¶](#)  
2 years ago

While waiting for native support for typed arrays, here are a couple of alternative ways to ensure strong typing of arrays by abusing variadic functions. The performance of these methods is a mystery to the writer and so the responsibility of benchmarking them falls unto the reader.

PHP 5.6 added the splat operator (...) which is used to unpack arrays to be used as function arguments. PHP 7.0 added scalar type hints. Latter versions of PHP have further improved the type system. With these additions and improvements, it is possible to have a decent support for typed arrays.

```
<?php
declare (strict_types=1);

function typeArrayNullInt(?int ...$arg): void {
}

function doSomething(array $ints): void {
    (function (?int ...$arg) {})(...$ints);
    // Alternatively,
    (fn (?int ...$arg) => $arg)(...$ints);
    // Or to avoid cluttering memory with too many closures
    typeArrayNullInt(...$ints);
```



```

/* ... */
}

function doSomethingElse(?int ...$ints): void {
/* ... */
}

$ints = [1,2,3,4,null];
doSomething ($ints);
doSomethingElse (...$ints);
?>

```

Both methods work with all type declarations. The key idea here is to have the functions throw a runtime error if they encounter a typing violation. The typing method used in `doSomethingElse` is cleaner of the two but it disallows having any other parameters after the variadic parameter. It also requires the call site to be aware of this typing implementation and unpack the array. The method used in `doSomething` is messier but it does not require the call site to be aware of the typing method as the unpacking is performed within the function. It is also less ambiguous as the `doSomethingElse` would also accept `n` individual parameters where as `doSomething` only accepts an array. `doSomething`'s method is also easier to strip away if native typed array support is ever added to PHP. Both of these methods only work for input parameters. An array return value type check would need to take place at the call site.

If `strict_types` is not enabled, it may be desirable to return the coerced scalar values from the type check function (e.g. floats and strings become integers) to ensure proper typing.

[up](#)  
[down](#)

11

[crash ¶](#)

**2 years ago**

The documentation lacks the information, that it's possible to change the return type of a method defined in an interface when the interface's methods return type is defined as ``mixed``.

From the RFC:

"The mixed return type could be narrowed in a subclass as this is covariant and is allowed in LSP." ([https://wiki.php.net/rfc/mixed\\_type\\_v2](https://wiki.php.net/rfc/mixed_type_v2))

This means the following code is valid in PHP 8.0:

```

<?php

interface ITest
{
    public function apfel(): mixed; // valid as of 8.0
}

class Test implements ITest
{
    public function apfel(): array // more explicit
    {
        return [];
    }
}

var_dump((new Test())->apfel());
?>

```

You can see the result here: <https://3v4l.org/PXDB6>

[+add a note](#)

- [Типы](#)
  - [Введение](#)
  - [Система типов](#)
  - [NULL](#)
  - [Логические значения](#)
  - [Целые числа](#)
  - [Числа с плавающей точкой](#)
  - [Строки](#)
  - [Числовые строки](#)
  - [Массивы](#)
  - [Объекты](#)
  - [Перечисления](#)
  - [Ресурсы](#)

- [Callable и callback-функции](#)
- [Mixed](#)
- [Void](#)
- [Never](#)
- [Относительные типы классов](#)
- [Типы значений](#)
- [Итерируемые значения](#)
- [Объявления типов](#)
- [Манипуляции с типами](#)
- [Copyright © 2001-2024 The PHP Group](#)
- [My PHP.net](#)
- [Contact](#)
- [Other PHP.net sites](#)
- [Privacy policy](#)

