АРГУМЕНТЫ ФУНКЦИЙ.
Передача аргументов по значению и по ссылке, аргументов по умолчанию и возвращаемые значения

Функция может принимать информацию в виде списка аргументов, который является списком разделенных запятыми выражений. Аргументы вычисляются слева направо.

PHP поддерживает передачу аргументов по значению (по умолчанию), передачу аргументов по ссылке, и значения по умолчанию.

Списки аргументов переменной длины также поддерживаются (смотрите также описания функций func_num_args(), func_get_arg() ифunc_get_args() для более детальной информации).

Пример 1. *Передача массива в функцию*

```
<?php
function takes_array($input)
{
   echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
?>
```

Передача аргументов по ссылке

По умолчанию аргументы в функцию передаются по значению (это означает, что если вы измените значение аргумента внутри функции, то вне ее значение все равно останется прежним). Если вы хотите разрешить функции модифицировать свои аргументы, вы должны передавать их по ссылке. Чтобы аргумент всегда передавался по ссылке, можно указать амперсанд (&) перед именем аргумента в описании функции:

Пример 2. *Передача аргументов по ссылке*

```
<?php
function add_some_extra(&$string)
{
   $string .= 'и кое-что еще.';
}
$str = 'Это строка, ';
add_some_extra($str);
echo $str;    // выведет 'Это строка, и кое-что еще.'
?>
```

Значения аргументов по умолчанию

Функция может определять значения по умолчанию в стиле C++ для скалярных аргументов, например:

Пример 3. *Использование значений по умолчанию в определении функции*

```
<?php
function makecoffee($type = "капучино")
{
   return "Готовим чашку $type.\n";
}
echo makecoffee();
echo makecoffee(null);
echo makecoffee("эспрессо");
?>
```

Результат выполнения данного примера:
Готовим чашку капучино.
Готовим чашку .
Готовим чашку эспрессо.

PHP также позволяет использовать массивы (array) и специальный тип **NULL** в качестве значений по умолчанию, например:

Пример 4. *Использование нескалярных типов в качестве значений по умолчанию*

```
<?php
function makecoffee($types = array("капуччино"), $coffeeMaker = NULL)
{
   $device = is_null($coffeeMaker) ? "вручную" : $coffeeMaker;
   return "Готовлю чашку ".join(", ", $types)." $device.\n";
}
echo makecoffee();
echo makecoffee(array("капуччино", "лавацца"), "в чайнике");
?>
```

Значение по умолчанию должно быть константным выражением, а не (к примеру) переменной или вызовом функции/метода класса. Аргументы, для которых установлены значения по умолчанию, должны находиться правее аргументов, для которых значения по умолчанию не заданы, в противном случае ваш код может работать не так, как вы этого ожидаете. Рассмотрим следующий пример:

Пример 5. *Некорректное использование значений по умолчанию*

```
<?php
function makeyogurt($type = "ацидофил", $flavour)
{
   return "Готовим чашку из бактерий $type со вкусом $flavour.\n";
}
 echo makeyogurt("малины");   // Не будет работать так, как мы могли бы ожидать
?>
```

Результат выполнения данного примера:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/phptest/functest.html on line 41
Готовим чашку из бактерий малины со вкусом.
```

Сравним его со следующим примером:

Пример 6. *Корректное использование значений по умолчанию*

```
<?php
function makeyogurt($flavour, $type = "ацидофил")
{
   return "Готовим чашку из бактерий $type со вкусом $flavour.\n";
}
echo makeyogurt("малины");   // отрабатывает правильно
?>
```

Результат выполнения данного примера:

```
Готовим чашку из бактерий ацидофил со вкусом малины.
```

Замечание: Начиная с PHP 5, значения по умолчанию могут быть переданы по ссылке.

Объявление типов

Объявления типов позволяют функциям строго задавать тип передаваемых параметров. Передача в функцию значений несоответствующего типа будет приводить к ошибке: в PHP 5 это будет обрабатываемая фатальная ошибка, а в PHP 7 будет выбрасываться исключение TypeError.

Чтобы объявить тип агрумента, необходимо перед его именем добавить имя требуемого типа. Также можно объявить тип **NULL**, чтобы указать, что значением по умолчанию аргумента является **NULL**.

Допустимые типы

Тип	*Описание*	*Минимальная версия PHP*
Имя класса/интерфейса	Агрумент должен быть *instanceof*, что и имя класса или интерфейса.	PHP 5.0.0
self	Этот параметр должен быть *instanceof* того же класса, в методе которого он указан. self можно использовать только в методах класса, либо объекта класса.	PHP 5.0.0
array	Аргумент должен быть типа array.	PHP 5.1.0

Тип	*Описание*	*Минимальная версия PHP*
callable	Аргумент должен быть корректным callable типом.	PHP 5.4.0
bool	Аргумент должен быть типа boolean.	PHP 7.0.0
float	Аргумент должен быть float типа.	PHP 7.0.0
int	Аргумент должен быть типа integer.	PHP 7.0.0
string	Аргумент должен иметь тип string.	PHP 7.0.0
iterable	Параметр должен быть либо массивом, либо экземпляром класса, реализующего Traversable.	PHP 7.1.0

Псевдонимы для вышеперечисленных скалярных типов не поддерживаются. Вместо этого они рассматриваются как имена классов или интерфейсов. К примеру, используя boolean как параметр или возвращаемое значение, потребует, чтобы эти аргумент или возвращаемое значение были instanceof класса или интерфейса boolean, а не типа bool:

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

Результат выполнения данного примера:

```
 Fatal error: Uncaught TypeError: Argument 1 passed to test() must be an instance of boolean, boolean given, called in - on line 1 and defined in -:1
```

Примеры

Пример 7. *Основные объявления типов-классов*

```
<?php
class C {}
class D extends C {}
// Это не является расширением класса C.
class E {}
function f(C $c) {
    echo get_class($c)."\n";
}
f(new C);
f(new D);
f(new E);
?>
```

Результат выполнения данного примера:

```
C
D
Fatal error: Uncaught TypeError: Argument 1 passed to f() must be an instance of C, instance of E given, called in - on line 14 and defined in -:8
Stack trace:
#0 -(14): f(Object(E))
#1 {main}
  thrown in - on line 8
```

Пример 8. *Основные объявления типов-интерфейсов*

```
<?php
interface I { public function f(); }
class C implements I { public function f() {} }
// Это не реализует интерфейс I.
class E {}
function f(I $i) {
    echo get_class($i)."\n";
}
f(new C);
f(new E);
?>
```

Результат выполнения данного примера:

```
C
Fatal error: Uncaught TypeError: Argument 1 passed to f() must implement interface I, instance of 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 9. *Объявление типа Null*

```
<?php
class C {}
function f(C $c = null) {
   var_dump($c);
}
f(new C);
f(null);
?>
```

Результат выполнения данного примера:

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

Строгая типизация

По умолчанию, PHP будет пытаться привести значения несоответствующих типов к скалярному типу, если это возможно. Например, если в функцию передается integer, а тип аргумента объявлен string, в итоге функция получит преобразованное string значение.

Для отдельных файлов можно включать режим строгой типизации. В этом режиме в функцию можно передавать значения только тех типов, которые объявлены для аргументов.

В противном случае будет выбрасываться исключение TypeError. Есть лишь одно исключение - integer можно передать в функцию, которая ожидает значение типа float. Вызовы функций внутри встроенных функций не будут затронуты директивой *strict_types*.

Для включения режима строгой типизации используется выражение *declare* в объявлении *strict_types*:

Внимание: Включение режима строгой типизации также повлияет на объявления типов возвращаемых значений.

Режим строгой типизации распространяется на вызовы функций совершенные из файла, в котором этот режим включен, а не на функции, которые в этом файле объявлены. Если файл без строгой типизации вызывает функцию, которая объявлена в файле с включенным режимом, значения аргументов будут приведены к нужным типам и ошибок не последует.

Строгая типизация применима только к скалярным типам и работает только в PHP 7.0.0 и выше. Равно как и сами объявления скалярных типов добавлены в этой версии.

Пример 10. *Строгая типизация*

```
<?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 данного примера:

```
int(3)
Fatal error: Uncaught TypeError: Argument 1 passed to sum() must be of the type integer, 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. *Слабая типизация*

```
<?php
function sum(int $a, int $b) {
   return $a + $b;
}
var_dump(sum(1, 2));
// Будут приведены к целым числам: обратите внимание на результат ниже!
var_dump(sum(1.5, 2.5));
?>
```

Результат выполнения данного примера:

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

Пример 12. *Обработка исключения TypeError*

```
<?php
declare(strict_types=1);
function sum(int $a, int $b) {
   return $a + $b;
}
try {
   var_dump(sum(1, 2));
   var_dump(sum(1.5, 2.5));
} catch (TypeError $e) {
   echo 'Ошибка: '.$e->getMessage();
}
?>
```

Результат выполнения данного примера:

```
int(3)
Error: Argument 1 passed to sum() must be of the type integer, float given, called in - on line 10
```

Списки аргументов переменной длины

PHP поддерживает списки аргументов переменной длины для функций, определяемых пользователем. Для версий PHP 5.6 и выше это делается добавлением многоточия (...). Для версий 5.5 и старше используются функции func_num_args(), func_get_arg() ифunc_get_args().

... в PHP 5.6+

В версиях PHP 5.6 и выше список аргументов может содержать многоточие ..., чтобы показать, что функция принимает переменное количество аргументов. Аргументы в этом случае будут переданы в виде массива. Например:

Пример 13. *Использование ... для доступа к аргументам*

```
<?php
function sum(...$numbers) {
   $acc = 0;
   foreach ($numbers as $n) {
      $acc += $n;
   }
   return $acc;
}
echo sum(1, 2, 3, 4);
?>
```

Результат выполнения данного примера:

```
10
```

Многоточие (...) можно использовать при вызове функции, чтобы распаковать массив (array) или Traversable переменную в список аргументов:

Пример 14. *Использование ... для передачи аргументов*

```
<?php
function add($a, $b) {
   return $a + $b;
}
 echo add(...[1, 2])."\n";
$a = [1, 2];
echo add(...$a);
?>
```

Результат выполнения данного примера:

```
3
3
```

Можно задать несколько аргументов в привычном виде, а затем добавить ....

Также можно добавить подсказку типа перед .... В этом случае PHP будет следить, чтобы все аргументы, обработанные многоточием (...) были того же типа, что указан в подсказке.

Пример 15. *Аргументы с подсказкой типов*

```
<?php
function total_intervals($unit, DateInterval ...$intervals) {
   $time = 0;
   foreach ($intervals as $interval) {
      $time += $interval->$unit;
   }
   return $time;
}
$a = new DateInterval('P1D');
$b = new DateInterval('P2D');
echo total_intervals('d', $a, $b).' days';
// Это не сработает, т.к. null не является объектом DateInterval.
echo total_intervals('d', null);
?>
```

Результат выполнения данного примера:

```
3 days
Catchable fatal error: Argument 2 passed to total_intervals() must be an instance of DateInterval, null given, called in - on line 14 and defined in - on line 2
```

Чтобы передавать аргументы по ссылке, перед ... нужно поставить амперсанд (&).

Предыдущие версии PHP

Для указания того, что функция принимает переменное число аргументов, никакой специальный синтаксис не используется. Для доступа к аргументам необходимо использовать функции func_num_args(), func_get_arg() и func_get_args().

В первом примере выше было показано, как задать список аргументов переменной длины для версий PHP 5.5 и более ранних:

Пример 16. *Доступ к аргументам в PHP 5.5 и ранних версий*

```
<?php
function sum() {
   $acc = 0;
   foreach (func_get_args() as $n) {
      $acc += $n;
   }
   return $acc;
}
echo sum(1, 2, 3, 4);
?>
```

Результат выполнения данного примера:

```
10
```