Функции

Функция ­ поименованный фрагмент программного кода (подпрограмма), к которому можно обратиться из другого места программы. С именем функции неразрывно связан адрес первой инструкции (оператора), входящей в функцию, которой передаётся управление при обращении к функции. После выполнения функции, управление возвращается обратно в адрес возврата — точку программы, где данная функция была вызвана.

Функция объявляется при помощи ключевого слова function​ ​, после которого следует имя функции, в круглых скобках параметры функции и в фигурных скобках записываются различные операторы, составляющие тело функции:

function MyFunction()

{

// операторы

}

Если функция возвращает какое-либо значение, в теле функции обязательно должен присутствовать оператор return:

function MyFunction()

{

// Вычисления

return $ret; // возвращается значение переменной $ret

}

Простой пример работы с функциями:

<?php

function get\_sum()

{

$sum = 10 + 5;

return $sum;

}

echo get\_sum(); // выводит 15

?>

Эта функция не принимает ни одного аргумента, а просто вычисляет сумму и возвращает полученный результат. После этого она вызывается в теле оператора echo​ ​для вывода результата в браузер. Модифицируем эту функцию так, чтобы она не возвращала полученный результат, а выводила его в браузер. Для этого достаточно внести оператор echo ​ ​в тело функции.

<?php

function get\_sum()

{

$sum = 10 + 5;

echo $sum;

}

get\_sum();

?>

function square($num) {

return $num \* $num;

// Этот код никогда не исполнится

echo "Мертвый код";

}

echo square(4); // выводит 16

Во многих языка программирования функция не может вызываться до ее объявления. В PHP отсутствуют подобные ограничения, функция может вызываться до ее объявления.

Это правило изменяется, если объявление функции осуществляется внутри фигурных скобок. Функции могут быть объявлены в блоке, обрамленном фигурными скобками.

<?php

// Объявляем логическую переменную

$flag = TRUE;

// Если переменная $flag равна TRUE, объявляем функцию

if ($flag) {

function get\_sum() {

$sum = 10 + 5;

echo $sum;

}

}

// Вызываем функцию, если переменная $flag равна TRUE

if ($flag) get\_sum(); // 15

?>

Однако объявить функцию позднее ее вывода в этом случае уже не получится.

<?php

// Объявляем логическую переменную

$flag = TRUE;

// Вызываем функцию, если переменная $flag равна TRUE

if ($flag) get\_sum(); // Ошибка

// Если переменная $flag равна TRUE, объявляем функцию

if ($flag) {

function get\_sum() {

$sum = 10 + 5;

echo $sum;

}

}

?>

Попытка вызова функции, объявленной условно, раньше объявления приводит к генерации ошибки.

#### 

#### Параметры функции

Можно значительно увеличить гибкость функции, если складываемые числа будут передаваться в качестве параметров.

<?php

function get\_sum($left, $right) {

$sum = $left + $right;

echo $sum;

}

get\_sum(10, 5); // 15

?>

Переменная, содержащая значение, переданное через аргумент, называется параметром функции, т. е. в примере числа 10 и 5 являются аргументами, а переменные $left и $right — параметрами.

В качестве параметров могут выступать выражения, и даже другие функции. В PHP выражения в этом случае вычисляются слева направо

<?php

function funct($left, $middle, $right) {

echo $left . "<br>";

echo $middle . "<br>";

echo $right . "<br>";

}

$i = 10;

funct(++$i, $i = $i \* 2, ­­$i);

?>

Результатом выполнения скрипта из будет такая последовательность чисел:

11   
22   
21

Передача параметров по значению и ссылке

В рассмотренных примерах аргументы функции передаются по значению, т. е. значения параметров изменяются только внутри функции, и эти изменения не влияют на значения переменных за пределами функции.

<?php

function get\_sum($var){ // аргумент передается по значению

$var = $var + 5;

return $var;

}

$new\_var = 20;

echo get\_sum($new\_var); // 25

echo "<br>$new\_var"; // 20

?>

Для того чтобы переменные, переданные функции, сохраняли свое значение при выходе из нее, применяется передача параметров по ссылке. Для этого перед именем переменной необходимо поместить амперсанд (​&​):

function get\_sum(&$var)

В этом случае переменная ​$var будет передана по ссылке. В случае, если аргумент передается по ссылке, при любом изменении значения параметра происходит изменение переменной аргумента

<?php

function get\_sum(&$var) // аргумент передается по ссылке

{

$var = $var + 5;

return $var;

}

$new\_var = 20;

echo get\_sum($new\_var); // выводит 25

echo "<br>$new\_var"; // выводит 25

?>

Необязательные параметры

Параметры можно объявлять как необязательные. Для этого при объявлении параметра необходимо присвоить ему значение по умолчанию.

<?php

function get\_sum($left = 10, $right = 5) {

$sum = $left + $right;

echo $sum;

}

get\_sum(); // выводит 15

get\_sum(5); // выводит 10

get\_sum(5, 0); // выводит 5

?>

Если функции get\_sum() не передаются параметры, она успешно производит вычисления с участием параметров по умолчанию. Если функция содержит множество обязательных и необязательных параметров, то все обязательные параметры следует располагать до необязательных. В примере ниже приводится некорректное объявление функции get\_sum(), в котором необязательный параметр предшествует обязательному.

<?php

function get\_sum($left = 10, $right)

{

$sum = $left + $right;

echo $sum;

}

get\_sum(5);

?>

Переменное количество параметров

С­подобные языки программирования обычно поддерживают синтаксис функций с переменным количеством параметров. В PHP для этого необходимо объявить функцию без параметров — такой функции можно передавать любое их количество, без каких-либо последствий.

<?php

function get\_sum() {

echo ‘Вызов функции’;

}

get\_sum(5, ‘Второй параметр’ 124);

?>

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.

function sum(...$numbers) {  
    // инструкция  
}

Можно убедиться, что интерпретатор PHP не генерирует ни замечаний, ни предупреждений (в противовес ситуации, когда функция имеет хотя бы один параметр). Для работы с переменным количеством параметров в PHP предусмотрены специальные функции.

|  |  |
| --- | --- |
| Функция | Описание |
| func\_num\_args() | Возвращает количество параметров, переданных функции |
| func\_get\_args() | Возвращает массив с параметрами, переданными функции |
| func\_get\_arg($arg\_num) | Возвращает значение параметра с номером $arg\_num. Если параметр с таким номером не существует, функция возвращает FALSE |

Ниже пример функции, которая выводит значения всех своих параметров.

<?php

// Объявляем функцию

function get\_parameters() {

for ($i = 0; $i < func\_num\_args(); $i++)

{

echo "Параметр номер $i: ".func\_get\_arg($i)."<br>";

}

}

// Вызываем функцию

echo get\_parameters("Hello,", "world", "!");

?>

Результатом выполнения скрипта будут следующие строки:

Параметр номер 0: Hello,

Параметр номер 1: world

Параметр номер 2: !

Глобальные переменные

Переменные в функциях имеют локальную область видимости. Это означает, что даже если локальная (внутри функции) и внешняя (вне функции) переменные имеют одинаковые имена, то изменение локальной переменной никак не повлияет на внешнюю переменную.

<?php

function get\_sum()

{

$var = 5; // локальная переменная

echo $var;

}

$var = 10; // внешняя переменная

get\_sum(); // выводит 5 (локальная переменная)

echo "<br>$var"; // выводит 10 (внешняя переменная)

?>

Локальную переменную можно сделать внешней, если перед ее именем указать ключевое слово global. В этом случае изменения как внутри функции, так и вне ее будут влиять на переменную, а сама переменная будет называться глобальной. Если локальная переменная объявлена как global, то к ней возможен доступ из любой части программы.

<?php

function get\_sum()

{

global $var;

$var = 5; // изменяем глобальную переменную

echo $var;

}

$var = 10;

echo "$var<br>"; // выводит 10

get\_sum(); // выводит 5(глобальная переменная изменена)

echo "$var<br>"; // выводит 5

?>

Доступ к глобальным переменным можно получить также через суперглобальный массив $GLOBALS.

<?php

function get\_sum()

{

$GLOBALS['var'] = 20; // изменяем глобальную переменную $var echo($GLOBALS['var']);

}

$var = 10;

echo "$var<br>"; // выводит 10

get\_sum(); // выводит 20 (глобальная переменная изменена)

?>

Массив $GLOBALS доступен в области видимости любой функции и содержит все глобальные переменные, которые используются в программе.

Статические переменные

Временем жизни переменной называется интервал выполнения программы, в течение которого она существует. Поскольку локальные переменные имеют своей областью видимости функцию, то время жизни локальной переменной определяется временем выполнения функции, в которой она объявлена. Это означает, что в разных функциях совершенно независимо друг от друга могут использоваться переменные с одинаковыми именами. Локальная переменная при каждом вызове функции инициализируется заново, поэтому функция­счетчик всегда будет возвращать значение 1.

<?php

function counter()

{

$counter = 0;

return ++$counter;

}

?>

Для того чтобы локальная переменная сохраняла свое предыдущее значение при новых вызовах функции, ее можно объявить статической при помощи ключевого слова static.

<?php

function counter()

{

static $counter = 0;

return ++$counter;

}

?>

В скрипте $counter устанавливается в ноль при первом вызове функции и при последующих вызовах функции помнит, каким было значение переменной при предыдущих вызовах.

Временем жизни статических и глобальных переменных является время выполнения сценария. То есть если пользователь перезагружает страницу, что приводит к новому выполнению сценария, переменная $counter инициализируется заново.

Возврат массива функцией

Функция может возвращать массив в качестве значения, для этого достаточно передать его в качестве параметра оператору return. Более того, такой массив может создаваться динамически при помощи конструкции array(). К такому приему прибегают всякий раз, когда функция должна вернуть несколько значений, а передача значений по ссылке не допускается.

Ниже демонстрируется функция format\_size(), которая принимает в качестве значения размер файла в байтах и возвращает массив, первый элемент которого содержит размер в байтах, второй — в килобайтах, третий — в мегабайтах, а четвертый — в гигабайтах.

<?php

function format\_size($byte)

{

$kbyte = $byte / 1024;

$mbyte = $kbyte / 1024;

$gbyte = $mbyte / 1024;

return array($byte, $kbyte, $mbyte, $gbyte);

}

?>

Оперировать массивом в скрипте не всегда удобно, особенно, если он имеет постоянное небольшое количество элементов. Поэтому часто при вызове функции массив сразу же разбивается на переменные при помощи конструкции list().

<?php

...

list($byte, $kbyte, $mbyte, $gbyte) = format\_size(18642678);

?>

Рекурсивные функции

Рекурсия — это вызов функцией самой себя, пример рекурсивной функции приведен ниже

<?php

function callself($counter) {

// Если параметр $counter больше, продолжаем рекурсивный спуск

if ($counter>0) {

// Уменьшаем значение параметра $counter и выводим его значение // в окно браузера

echo($counter­­)."<br>";

// Осуществляем рекурсивный вызов функции callself()

Callself($counter);

}

// Если значение параметра меньше или равно 0, прекращаем

// рекурсивный спуск

else return;

}

// Вызываем функцию callself()

callself(4);

?>

Результатом работы функции будет последовательность цифр:

4

3

2

1

Функция callself() вызывает саму себя до тех пор, пока ее параметр $counter положителен и не равен нулю. Рекурсивных функций по возможности стараются избегать, т.к. они относятся к трудным по восприятию конструкциям языка, и отладка их достаточно сложна, особенно, когда приходится иметь дело не с простейшей рекурсивной функцией, представленной в выше, а со сложной функцией, осуществляющей рекурсивный вызов в нескольких местах функции.

Опасность использования неотлаженных рекурсивных функций заключается в возможности перехода их в режим бесконечной рекурсии, когда условие, прекращающее спуск вниз по рекурсии из-за ошибки, не наступает, в результате чего, как и в случае бесконечных циклов, наступает зависание программы.

Практически в любом случае можно избежать рекурсивных функций. Исключение составляют задачи, так или иначе связанные с обходом деревьев. К таким задачам относится, например, удаление каталогов, когда число файлов и подкаталогов заранее не известно и необходимо вызывать функцию удаления до тех пор, пока не будут удаленны файлы на самом глубоком уровне вложенности.

Вложенные функции

Язык программирования PHP позволяет объявлять функции внутри другой функции. В отличие от обычных функций, вложенная функция не может использоваться до тех пор, пока не будет осуществлен вызов основной функции, который произведет объявление вложенной.

<?php

// Объявление внешней и вложенной функций

function outter() {

function inner() {

echo "Hello, world!";

}

}

// Вызываем функцию outter(), чтобы объявить функцию inner()

outter();

// Функция inner() не может быть вызвана до тех пор,

// пока не будет вызвана функция outter();

inner();

?>

Динамическое имя функции

По аналогии с переменными, имя функции может быть динамическим и храниться в строковой переменной — передача такой переменной оператором круглых скобок (с параметрами, если они требуются) приводит к вызову функции.

<?php

// Объявление функций

function hello() {

echo "Hello!";

}

function bye() {

echo "Bye!";

}

// Случайный выбор функции

if (rand(0, 1)) $var = "hello";

else $var = "bye";

// Вызов функции

$var();

?>

Анонимные функции

Анонимные функции, также известные как замыкания (closures​ ​), позволяют создавать функции, не имеющие определенных имен. Они наиболее полезны в качестве значений callback­параметров, но также могут иметь и множество других применений.

<?php

$greet = function($name) {

printf("Hello %s<br />", $name);

};

$greet('World');

$greet('PHP');

Замыкания могут также наследовать переменные из родительской области видимости. Любая подобная переменная должна быть передана в языковую конструкцию use​​.

<?php

$message = 'hello';

// Без "use"

$example = function () {

var\_dump($message);

};

echo $example();

// Наследование $message

$example = function () use ($message) {

var\_dump($message);

};

echo $example();

// Наследуемое значение переменной определяется в момент создания функции, // а не в момент ее вызова $message = 'world';

echo $example();

// Сброс $message

$message = 'hello';

// Наследование по ссылке

$example = function () use (&$message) {

var\_dump($message);

};

echo $example();

// Измененное значение в родительской области видимости

// отражается и внутрь функции $message = 'world';

echo $example();

// Замыканию также доступны и обычные аргументы

$example = function ($arg) use ($message) {

var\_dump($arg . ' ' . $message);

};

$example("hello");

?>

Проверка существования функции

Функция с заданным или динамическим именем может быть доступна для вызова или нет. В связи с этим одной из важных задач является проверка существования функции. Эту задачу выполняют функции, представленные в таблице.

|  |  |
| --- | --- |
| Функция | Описание |
| function\_exists($function\_name) | Возвращает TRUE, если функция с именем $function\_name существует, в противном случае возвращается FALSE |
| get\_defined\_functions() | Возвращает массив всех доступных в настоящий момент функций |
| get\_extension\_funcs($module\_name) | Возвращает массив с именами функций для расширения $module\_name |
| extension\_loaded($name) | Проверяет, подключено ли расширение с именем $name |
| get\_loaded\_extensions ([$zend\_extensions = FALSE]) | Возвращает массив подключенных расширений |

Проверить существование одиночной функции проще всего при помощи функции function\_exists()

<?php

if (TRUE) {

function get\_sum\_if($fst, $snd) {

return $fst + $snd;

}

}

if (function\_exists("get\_sum")) // TRUE

echo "Функция get\_sum() существует<br>";

else

echo "Функция get\_sum() не существует<br>";

if (function\_exists("get\_sum\_if")) // TRUE

echo "Функция get\_sum\_if () существует<br>";

else

echo "Функция get\_sum\_if () не существует<br>";

if (function\_exists("get\_sum\_if\_post")) // FALSE

echo "Функция get\_sum\_if\_post() существует<br>";

else

echo "Функция get\_sum\_if\_post() не существует<br>";

function get\_sum($fst, $snd) {

return $fst + $snd;

}

if (TRUE) {

function get\_sum\_if\_post($fst, $snd) {

return $fst + $snd;

}

}

?>

Результатом выполнения скрипта будут следующие строки:

Функция get\_sum() существует

Функция get\_sum\_if () существует

Функция get\_sum\_if\_post() не существует

При помощи функции get\_defined\_functions()​ ​ можно получить массив всех функций, доступных на текущий момент

<?php

function get\_sum($fst, $snd) {

return $fst + $snd;

}

$arr = get\_defined\_functions();

echo "<pre>";

print\_r($arr);

echo "</pre>";

?>

Функция get\_defined\_functions()​ возвращает двумерный ассоциативный массив, первый элемент internal содержит предопределенные функции из ядра PHP и расширений, второй элемент user содержит функции, определенные пользователем:

Array (

[internal] => Array (

=> zend\_version   
=> func\_num\_args   
=> func\_get\_arg   
=> func\_get\_args   
=> strlen   
=> strcmp   
=> strncmp   
...   
=> socket\_getopt   
=> socket\_setopt

)

[user] => Array (   
 [0] => get\_sum

)

)

Функция ​get\_extension\_funcs() позволяет вернуть массив с именами функций, принадлежащих расширению, имя которого передается в качестве параметра. Приведенный ниже скрипт возвращает список функция, которые определены в расширении GD.

<?php

// Получаем массив с именами функций для работы с MySQL

$arr = get\_extension\_funcs("gd"); echo "<pre>"; print\_r($arr); echo "</pre>";

?>

Результатом выполнения скрипта будет следующий дамп массива $arr:

Array (

=> gd\_info   
=> imagearc   
=> imageellipse   
=> imagechar   
...   
=> imagecolormatch   
=> imagefilter   
=> imageconvolution

)

Для того чтобы функция ​get\_extension\_funcs() вернула результат, соответствующее расширение должно быть подключено в конфигурационном файле php.ini. Если расширение не подключено, функция вернет FALSE. Для проверки факта, подключено расширение или нет, предназначена специальная функция extension\_loaded(), которая принимает в качестве параметра имя расширения и возвращает TRUE, если оно подключено, и FALSE в противном случае.

Вспомогательные функции

Ниже приведен список часто используемых вспомогательных функций, которые содержатся в ядре PHP.

|  |  |
| --- | --- |
| Функция | Описание |
| exit([$status]) | Прекращает выполнение скрипта. Если указан необязательный параметр $status, он выводится в окно браузера |
| die() | Синоним для функции exit() |
| sleep($seconds) | Осуществляет задержку выполнения скрипта на $seconds секунд |
| time\_nanosleep($seconds, $nanoseconds) | Осуществляет задержку выполнения скрипта на $seconds секунд и $nanoseconds наносекунд |
| usleep($micro\_seconds) | Осуществляет задержку выполнения скрипта на $micro\_seconds микросекунд |
| eval($code) | Выполняет PHP­код из строки ​$code |
| empty($var) | Определяет, не пуста ли переменная $var, возвращая TRUE, если переменная пустая, и FALSE в противном |
| isset($var) | Определяет, существует ли переменная $var, возвращая TRUE, если переменная существует, и FALSE в противном случае |
| unset($var [, $var1 [, ...]] | Уничтожает одну или более переменных, массивов, объектов, переданных конструкции в качестве параметров |
| print\_r($var [, $return]) | Выводит дамп переменной, массива или объекта $var. Если необязательный параметр $return принимает значение TRUE, вместо вывода дампа в окно браузера, он возвращается в виде строки |
| var\_dump($var [, $var1 [, ...]]) | Выводит дамп одной и большего количества переменных, включая их типы данных |
| phpinfo() | Выводит таблицы с параметрами текущей версии PHP, расширений и переменных окружений |
| phpversion([$extension]) | Возвращает текущую версию PHP или версию расширения, если необязательный параметр  $extension содержит его имя |
| php\_uname([$mode]) | Возвращает сведения об операционной системе, в которой выполняется PHP. Необязательный параметр $mode может принимать значения параметров для одноименной утилиты UNIX­подобных операционных систем |
| getenv($varname) | Возвращает значение переменной окружения с именем $varname. |

Задание №1

Функция должна принимать массив строк и выводить каждую строку в отдельном параграфе.

Задание №2

Функция должна принимать 2 параметра: а) массив чисел; б) строку, обозначающую арифметическое действие, которое нужно выполнить со всеми элементами массива. Функция должна вывести результат.

Например: имя функции someFunction(array(1,2,3,4), ‘ – ’) Результат: 1 – 2 – 3 – 4 = 8

Задание №3

Функция должна принимать переменное число аргументов, но первым аргументом обязательно должна быть строка, обозначающая арифметическое действие, которое необходимо выполнить со всеми передаваемыми аргументами. Остальные аргументы целые и/или вещественные.

Например: имя функции someFunction(‘+’, 1, 2, 3, 5.2); Результат: 1 + 2 + 3 + 5.2 = 11.2

Задание №4

Функция должна принимать два параметра – целые числа. Если в функцию переданы не целые числа, то функция должна выводить ошибку на экран и завершать работу. Если в функцию передали 2 целых числа, то функция должна отобразить таблицу умножения размером со значения параметров, переданных в функцию.

Например: tabl(4,3), то функция должна нарисовать следующий результат:

2 3 4

4 6 8

6 9 12

Задание №5

Функция должна принимать в качестве аргумента массив чисел и возвращать так же массив, но отсортированный по возрастанию.

Пример: В функцию передали [1, 22, 5, 66, 3, 57]. Вернула: [1, 3, 5, 22, 57, 66]

Задание №6

Создать рекурсивную функцию, которая принимает два целых числа, начальное и конечное значения, диапазон. Функция должна вывести список нечётных чисел в заданном диапазоне.

Например: В функцию передали: 10 и 35. Функция должна вывести список нечётных чисел в диапазоне от 10 до 35.

Задание №7

Функция принимает 1 строковый параметр и возвращает TRUE, если строка является палиндромом, FALSE в противном случае.

Палиндром – строка, одинаково читающаяся в обоих направлениях.

Задание №8

Создать 3 функции, принимающие по 1 строковому параметру и выводящие на экран разные фразы с переданным параметром.

Создать 4ю функцию, которая будет принимать 2 параметра:

а) целое число

б) строку и в зависимости от переданного числа функция должна вызвать одну из трёх функций со строковым параметром (1 – первая ф-ия, 2 – вторая, 3 -третья).

Четвёртая функция также должна обрабатывать некорректный ввод целого числа