



- [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

[Синтаксис генераторов »](#)  
[« Генераторы](#)

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

Change language: Russian

# Знакомство с генераторами

(PHP 5 >= 5.5.0, PHP 7, PHP 8)

Генераторы предоставляют лёгкий способ реализации простых [итераторов](#) без использования дополнительных ресурсов или сложностей, связанных с реализацией класса, реализующего интерфейс [Iterator](#).

Генератор позволяет вам писать код, использующий [foreach](#) для перебора набора данных без необходимости создания массива в памяти, что может привести к превышению лимита памяти, либо потребует довольно много времени для его создания. Вместо этого, вы можете написать функцию-генератор, которая, по сути, является обычной [функцией](#), за исключением того, что вместо [возврата](#) единственного значения, генератор может возвращать ([yield](#)) столько раз, сколько необходимо для генерации значений, позволяющих перебрать исходный набор данных.

Наглядным примером вышесказанного может послужить использование функции [range\(\)](#) как генератора. Стандартная функция [range\(\)](#) генерирует массив, состоящий из значений, и возвращает его, что может привести к генерации огромных массивов данных. Например, вызов **range(0, 1000000)** приведёт к использованию более 100 МБ оперативной памяти.

В качестве альтернативы мы можем создать генератор xrange(), который использует память только для создания объекта [Iterator](#) и сохранения текущего состояния, что потребует не больше 1 килобайта памяти.

## Пример #1 Реализация [range\(\)](#) как генератора

```
<?php
function xrange($start, $limit, $step = 1) {
    if ($start <= $limit) {
        if ($step <= 0) {
            throw new LogicException('Шаг должен быть положительным');
        }

        for ($i = $start; $i <= $limit; $i += $step) {
            yield $i;
        }
    } else {
        if ($step >= 0) {
            throw new LogicException('Шаг должен быть отрицательным');
        }

        for ($i = $start; $i >= $limit; $i += $step) {
            yield $i;
        }
    }
}

/* Обратите внимание, что и range() и xrange() дадут один и тот же вывод */

echo 'Нечётные однозначные числа с помощью range(): ';
foreach (range(1, 9, 2) as $number) {
    echo "$number ";
}
echo "\n";

echo 'Нечётные однозначные числа с помощью xrange(): ';
foreach (xrange(1, 9, 2) as $number) {
    echo "$number ";
}
?>
```

Результат выполнения приведённого примера:

```
Нечётные однозначные числа с помощью range():  1 3 5 7 9
Нечётные однозначные числа с помощью xrange(): 1 3 5 7 9
```

## Объект [Generator](#)

Когда функция генератор вызывается, она вернёт объект встроенного класса [Generator](#). Этот объект реализует интерфейс [Iterator](#), станет однонаправленным объектом итератора и предоставит методы, с помощью которых можно управлять его состоянием, включая передачу в него и возвращения из него значений.

[+add a note](#)

### User Contributed Notes 8 notes

[up](#)

[down](#)

173

[bloodjazman at gmail dot com ¶](#)

10 years ago

for the protection from the leaking of resources

see RFC [https://wiki.php.net/rfc/generators#closing\\_a\\_generator](https://wiki.php.net/rfc/generators#closing_a_generator)

and use finnaly

sample code

```
function getLines($file) {  
    $f = fopen($file, 'r');  
    try {  
        while ($line = fgets($f)) {  
            yield $line;  
        }  
    } finally {  
        fclose($f);  
    }  
}
```

```
foreach (getLines("file.txt") as $n => $line) {  
    if ($n > 5) break;  
    echo $line;  
}
```

[up](#)

[down](#)

40

[montoriusz at gmail dot com ¶](#)

7 years ago

Bear in mind that execution of a generator function is postponed until iteration over its result (the Generator object) begins. This might confuse one if the result of a generator is assigned to a variable instead of immediate iteration.

```
<?php
```

```
$some_state = 'initial';
```

```
function gen() {  
    global $some_state;
```

```
    echo "gen() execution start\n";  
    $some_state = "changed";
```

```
    yield 1;  
    yield 2;  
}
```

```
function peek_state() {  
    global $some_state;  
    echo "\$some_state = $some_state\n";
```

```

}

echo "calling gen()...\n";
$result = gen();
echo "gen() was called\n";

peek_state();

echo "iterating...\n";
foreach ($result as $val) {
echo "iteration: $val\n";
peek_state();
}

?>

```

If you need to perform some action when the function is called and before the result is used, you'll have to wrap your generator in another function.

```

<?php
/**
 * @return Generator
 */
function some_generator() {
global $some_state;

$some_state = "changed";
return gen();
}

?>

```

[up](#)

[down](#)

27

[info at boukeversteegh dot nl ¶](#)

**8 years ago**

Here's how to detect loop breaks, and how to handle or cleanup after an interruption.

```

<?php
function generator()
{
$complete = false;
try {

while (($result = some_function())) {
yield $result;
}

$complete = true;

} finally {
if (!$complete) {
// cleanup when loop breaks
} else {
// cleanup when loop completes
}
}

// Do something only after loop completes
}

?>

```

[up](#)

[down](#)

9

[chung1905 at gmail dot com ¶](#)

**4 years ago**

In addition to the note of "montoriusz at gmail dot com":

<https://www.php.net/manual/en/language.generators.overview.php#119275>

"If you need to perform some action when the function is called and before the result is used, you'll have to wrap your generator in another function."

You can use `Generator::rewind` instead (<https://www.php.net/manual/en/generator.rewind.php>)

Sample code:

```
<?php
```

```
/** function/generator definition */
```

```
echo "calling gen()...\n";
```

```
$result = gen();
```

```
$result->rewind();
```

```
echo "gen() was called\n";
```

```
/** iteration */
```

```
?>
```

[up](#)

[down](#)

18

[lubaev ¶](#)

**10 years ago**

Abstract test.

```
<?php
```

```
$start_time=microtime(true);
```

```
$array = array();
```

```
$result = '';
```

```
for($count=1000000; $count--;) 
```

```
{
```

```
$array[]=$count/2;
```

```
}
```

```
foreach($array as $val)
```

```
{
```

```
$val += 145.56;
```

```
$result .= $val;
```

```
}
```

```
$end_time=microtime(true);
```

```
echo "time: ", bcsub($end_time, $start_time, 4), "\n";
```

```
echo "memory (byte): ", memory_get_peak_usage(true), "\n";
```

```
?>
```

```
<?php
```

```
$start_time=microtime(true);
```

```
$result = '';
```

```
function it()
```

```
{
```

```
for($count=1000000; $count--;) 
```

```
{
```

```
yield $count/2;
```

```
}
```

```
}
```

```
foreach(it() as $val)
```

```
{
```

```
$val += 145.56;
```

```
$result .= $val;
```

```
}
$end_time=microtime(true);

echo "time: ", bcsub($end_time, $start_time, 4), "\n";
echo "memory (byte): ", memory_get_peak_usage(true), "\n";
```

```
?>
Result:
-----
| time | memory, mb |
-----
| not gen | 2.1216 | 89.25 |
|-----|
| with gen | 6.1963 | 8.75 |
|-----|
| diff | < 192% | > 90% |
-----
```

[up](#)  
[down](#)  
13  
[dc at libertyskull dot com ¶](#)

**9 years ago**  
Same example, different results:

```
-----
| time | memory, mb |
-----
| not gen | 0.7589 | 146.75 |
|-----|
| with gen | 0.7469 | 8.75 |
|-----|
```

Time in results varying from 6.5 to 7.8 on both examples.  
So no real drawbacks concerning processing speed.

[up](#)  
[down](#)  
-10  
[youssefbenhssaien at gmail dot com ¶](#)

**6 years ago**

A simple function to parse an ini configuration file

```
<?php
function parse_ini($file_path){
if(!file_exists($file_path)){
throw new Exception("File not exists ${file_path}");
}
$text = fopen($file_path, 'r');
while($line=fgets($text)){
list($key, $param) = explode('=', $line);
yield $key => $param;
}
}
?>

//Usage : parse_ini('param.ini') // returns Generator Object
//Usage : iterator_to_array(parse_ini('param.ini')); // returns an array
```

[up](#)  
[down](#)  
-33  
[Anonymous ¶](#)

**4 years ago**  
Same example, different results:

```
-----
```

time	memory, mb
not gen	0.7589   146.75
with gen	0.7469   8.75

Time in results varying from 6.5 to 7.8 on both exassmples.  
So no real drawbacks concerning processing speed.

[+add a note](#)

- [Генераторы](#)
  - [Знакомство с генераторами](#)
  - [Синтаксис генераторов](#)
  - [Сравнение генераторов с объектами класса Iterator](#)
- [Copyright © 2001-2024 The PHP Group](#)
- [My PHP.net](#)
- [Contact](#)
- [Other PHP.net sites](#)
- [Privacy policy](#)

