



- [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

[declare »](#)

[« switch](#)

- [Руководство по PHP](#)
- [Справочник языка](#)
- [Управляющие конструкции](#)

Change language: Russian ▾

match

(PHP 8)

Выражение `match` предназначено для ветвления потока исполнения на основании проверки совпадения значения с заданным условием. Аналогично оператору `switch`, выражение `match` принимает на вход выражение, которое сравнивается с множеством альтернатив. Но, в отличие от `switch`, оно обрабатывает значение в стиле, больше похожем на тернарный оператор. Также, в отличие от `switch`, используется строгое сравнение (`===`), а не слабое (`==`). Выражение `match` доступно начиная с PHP 8.0.0.

Пример #1 Структура выражения `match`

```
<?php
$return_value = match (subject_expression) {
    single_conditional_expression => return_expression,
    conditional_expression1, conditional_expression2 => return_expression,
};
?>
```

Пример #2 Простой пример использования `match`

```
<?php
$food = 'cake';
$return_value = match ($food) {
    'apple' => 'На столе лежит яблоко',
    'banana' => 'На столе лежит банан',
    'cake' => 'На столе стоит торт',
};
var_dump($return_value);
?>
```

Результат выполнения приведённого примера:

```
string(35) "На столе стоит торт"
```

Замечание: Результат `match` использовать не обязательно.

Замечание: Выражение `match` *должно* завершаться точкой с запятой `;`.

Выражение `match` похоже на оператор `switch` за исключением некоторых ключевых отличий:

- В отличие от `switch`, в `match` используется строгое сравнение (`===`).
- Выражение `match` возвращает результат.
- В `match` исполняется только одна, первая подошедшая, ветвь кода, тогда как в `switch` происходит сквозное исполнение начиная с подошедшего условия и до первого встретившегося оператора `break`.
- Выражение `match` должно быть исчерпывающим.

Также как и оператор `switch`, `match` последовательно проводит проверки на совпадение с заданными условиями. Выполнение кода условий происходит лениво, т.е. код следующего условия выполняется только если все предыдущие проверки провалились. Будет выполнена только одна ветвь кода, соответствующая подошедшему условию. Пример:

```
<?php
$result = match ($x) {
    foo() => ...,
    $this->bar() => ..., // $this->bar() не будет выполнен, если foo() === $x
    $this->baz => beep(), // beep() будет выполнен только если $x === $this->baz
    // etc.
};
?>
```

Условия в `match` могут быть множественными. В этом случае их следует разделять запятыми. Множественные условия работают по принципу логического ИЛИ и, по сути, являются сокращённой формой для случаев, когда несколько условий должны обрабатываться идентично.

```
<?php
$result = match ($x) {
    // Множественное условие:
    $a, $b, $c => 5,
    // Аналогично трём одиночным:
    $a => 5,
    $b => 5,
    $c => 5,
};
?>
```

Также можно использовать шаблон default. Этот шаблон совпадает с чем угодно, для чего не нашлось совпадений раньше. К примеру:

```
<?php
$expressionResult = match ($condition) {
    1, 2 => foo(),
    3, 4 => bar(),
    default => baz(),
};
?>
```

Замечание: Использование нескольких шаблонов default приведёт к фатальной ошибке `E_FATAL_ERROR`.

Выражение `match` должно быть исчерпывающим. Если проверяемое выражение не совпало ни с одним из условий, то будет выброшено исключение [UnhandledMatchError](#).

Пример #3 Пример необработанного выражения

```
<?php
$condition = 5;

try {
    match ($condition) {
        1, 2 => foo(),
        3, 4 => bar(),
    };
} catch (\UnhandledMatchError $e) {
    var_dump($e);
}
?>
```

Результат выполнения приведённого примера:

```
object(UnhandledMatchError)#1 (7) {
    ["message":protected]=>
    string(33) "Unhandled match value of type int"
    ["string":"Error":private]=>
    string(0) ""
    ["code":protected]=>
    int(0)
    ["file":protected]=>
    string(9) "/in/ICgGK"
    ["line":protected]=>
    int(6)
    ["trace":"Error":private]=>
    array(0) {
    }
    ["previous":"Error":private]=>
    NULL
}
```

Использование match для проверки сложных условий

Выражение `match` можно использовать не только для проверки идентичности, но и для любых выражений, возвращающих логическое значение. В этом случае в качестве входного параметра передаётся выражение `true`.

Пример #4 Использование match для ветвления в зависимости от вхождения в диапазоны целых чисел

```
<?php

$age = 23;

$result = match (true) {
    $age >= 65 => 'пожилой',
    $age >= 25 => 'взрослый',
    $age >= 18 => 'совершеннолетний',
    default => 'ребёнок',
};

var_dump($result);
?>
```

Результат выполнения приведённого примера:

```
string(11) "совершеннолетний"
```

Пример #5 Использование match для ветвления в зависимости от содержимого строки

```
<?php

$text = 'Bienvenue chez nous';

$result = match (true) {
    str_contains($text, 'Welcome') || str_contains($text, 'Hello') => 'en',
    str_contains($text, 'Bienvenue') || str_contains($text, 'Bonjour') => 'fr',
    // ...
};

var_dump($result);
?>
```

Результат выполнения приведённого примера:

```
string(2) "fr"
```

[+add a note](#)

User Contributed Notes 8 notes

[up](#)

[down](#)

72

[darius dot restivan at gmail dot com ¶](#)

2 years ago

This will allow for a nicer FizzBuzz solution:

```
<?php

function fizzbuzz($num) {
    print match (0) {
        $num % 15 => "FizzBuzz" . PHP_EOL,
        $num % 3  => "Fizz"   . PHP_EOL,
        $num % 5  => "Buzz"   . PHP_EOL,
        default => $num . PHP_EOL,
    };
}

for ($i = 0; $i <=100; $i++)
{
    fizzbuzz($i);
}
```

[up](#)

[down](#)

61

[Anonymous ¶](#)

2 years ago

```
<?php
```

```
function days_in_month(string $month, $year): int
{
    return match(strtolower(substr($month, 0, 3))) {
        'jan' => 31,
        'feb' => is_leap($year) ? 29 : 28,
        'mar' => 31,
        'apr' => 30,
        'may' => 31,
        'jun' => 30,
        'jul' => 31,
        'aug' => 31,
        'sep' => 30,
        'oct' => 31,
        'nov' => 30,
        'dec' => 31,
        default => throw new InvalidArgumentException("Bogus month"),
    };
}
```

can be more concisely written as

```
<?php
```

```
function days_in_month(string $month, $year): int
{
    return match(strtolower(substr($month, 0, 3))) {
        'apr', 'jun', 'sep', 'nov' => 30,
        'jan', 'mar', 'may', 'jul', 'aug', 'oct', 'dec' => 31,
        'feb' => is_leap($year) ? 29 : 28,
        default => throw new InvalidArgumentException("Bogus month"),
    };
}
```

```
?>
```

[up](#)

[down](#)

55

[Hayley Watson ¶](#)

3 years ago

As well as being similar to a switch, match expressions can be thought of as enhanced lookup tables – for when a simple array lookup isn't enough without extra handling of edge cases, but a full switch statement would be overweight.

For a familiar example, the following

```
<?php
```

```
function days_in_month(string $month): int
{
    static $lookup = [
        'jan' => 31,
        'feb' => 0,
        'mar' => 31,
        'apr' => 30,
        'may' => 31,
        'jun' => 30,
        'jul' => 31,
        'aug' => 31,
        'sep' => 30,
        'oct' => 31,
```

```

'nov' => 30,
'dec' => 31
];

$name = strtolower(substr($name, 0, 3));

if(isset($lookup[$name])) {
    if($name == 'feb') {
        return is_leap($year) ? 29 : 28;
    } else {
        return $lookup[$name];
    }
}

throw new InvalidArgumentException("Bogus month");
}

?>

```

with the fiddly stuff at the end, can be replaced by

```

<?php
function days_in_month(string $month): int
{
    return match(strtolower(substr($month, 0, 3))) {
        'jan' => 31,
        'feb' => is_leap($year) ? 29 : 28,
        'mar' => 31,
        'apr' => 30,
        'may' => 31,
        'jun' => 30,
        'jul' => 31,
        'aug' => 31,
        'sep' => 30,
        'oct' => 31,
        'nov' => 30,
        'dec' => 31,
        default => throw new InvalidArgumentException("Bogus month"),
    };
}

?>

```

Which also takes advantage of "throw" being handled as of PHP 8.0 as an expression instead of a statement.

[up](#)

[down](#)

5

[Sbastien ¶](#)

1 year ago

I use match instead of storing PDOStatement::rowCount() result and chaining if/elseif conditions or use the ugly switch/break :

```

<?php

$sql = <<<SQL
INSERT INTO ...
ON DUPLICATE KEY UPDATE ...
SQL;

$upkeep = $pdo->prepare($sql);

$count_untouched = 0;
$count_inserted = 0;
$count_updated = 0;

```

```
foreach ($data as $record) {
$upkeep->execute($record);
match ($upkeep->rowCount()) {
0 => $count_untouched++,
1 => $count_inserted++,
2 => $count_updated++,
};
}
```

```
echo "Untouched rows : {$count_untouched}\r\n";
echo "Inserted rows : {$count_inserted}\r\n";
echo "Updated rows : {$count_updated}\r\n";
```

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

6

[thomas at zuschneid dot de ¶](#)

11 months ago

While match allows chaining multiple conditions with ",", like:

```
<?php
$result = match ($source) {
cond1, cond2 => val1,
default => val2
};
?>
```

it seems not valid to chain conditions with default, like:

```
<?php
$result = match ($source) {
cond1 => val1,
cond2, default => val2
};
?>
```

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

8

[php at joren dot dev ¶](#)

1 year ago

If you want to execute multiple return expressions when matching a conditional expression, you can do so by stating all return expressions inside an array.

```
<?php
$countries = ['Belgium', 'Netherlands'];
$spoken_languages = [
'Dutch' => false,
'French' => false,
'German' => false,
'English' => false,
];
```

```
foreach ($countries as $country) {
match($country) {
'Belgium' => [
$spoken_languages['Dutch'] = true,
$spoken_languages['French'] = true,
$spoken_languages['German'] = true,
],
'Netherlands' => $spoken_languages['Dutch'] = true,
'Germany' => $spoken_languages['German'] = true,
'United Kingdom' => $spoken_languages['English'] = true,
};
}
```



```
var_export($spoken_languages);  
// array ( 'Dutch' => true, 'French' => true, 'German' => true, 'English' => false, )
```

?>

[up](#)

[down](#)

5

[mark at mannngo dot net ¶](#)

2 years ago

While you can't polyfill a language construct, you can mimic the basic behaviour with a simple array.

Using example 2 above:

```
<?php  
$food = 'apple';  
$return_value = match ($food) {  
    'apple' => 'This food is an apple',  
    'bar' => 'This food is a bar',  
    'cake' => 'This food is a cake',  
};  
print $return_value;  
?>
```

... you can get something similar with:

```
<?php  
$food = 'apple';  
$return_value = [  
    'apple' => 'This food is an apple',  
    'bar' => 'This food is a bar',  
    'cake' => 'This food is a cake',  
][$food];  
print $return_value;  
?>
```

[up](#)

[down](#)

0

[tm ¶](#)

2 years ago

If you are using a match expression for non-identity checks as described above make sure whatever you are using is actually returning `true` on success.

Quite often you rely on truthy vs. falsy when using if conditions and that will not work for match (for example `preg_match`). Casting to bool will solve this issue.

[+add a note](#)

- [Управляющие конструкции](#)
 - [Введение](#)
 - [if](#)
 - [else](#)
 - [elseif/else if](#)
 - [Альтернативный синтаксис управляющих структур](#)
 - [while](#)
 - [do-while](#)
 - [for](#)
 - [foreach](#)
 - [break](#)
 - [continue](#)
 - [switch](#)
 - [match](#)
 - [declare](#)
 - [return](#)

- [require](#)
- [include](#)
- [require_once](#)
- [include_once](#)
- [goto](#)
- [Copyright © 2001-2024 The PHP Group](#)
- [My PHP.net](#)
- [Contact](#)
- [Other PHP.net sites](#)
- [Privacy policy](#)

