




- [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

[match »](#)
[« continue](#)

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

Change language: Russian ▾

switch

(PHP 4, PHP 5, PHP 7, PHP 8)

Оператор `switch` похож на ряд операторов `IF` с одинаковым условием. Во многих случаях вам может понадобиться сравнивать одну и ту же переменную (или выражение) с множеством различных значений и выполнять различные участки кода в зависимости от того, какое значение принимает эта переменная (или выражение). Это именно тот случай, для которого удобен оператор `switch`.

Замечание: Обратите внимание, что в отличие от некоторых других языков, оператор [continue](#) применяется в конструкциях `switch` и действует подобно оператору `break`. Если у вас конструкция `switch` находится внутри цикла, и вам необходимо перейти к следующей итерации цикла, используйте `continue 2`.

Замечание:

Заметьте, что конструкция `switch/case` использует [нестрогое сравнение \(==\)](#).

В следующем примере каждый блок кода эквивалентен. В одном используется серия операторов `if` и `elseif`, а в другом - оператор `switch`. В каждом случае результат один и тот же.

Пример #1 Оператор switch

```
<?php
// Оператор switch:

switch ($i) {
case 0:
echo "i равно 0";
break;
case 1:
echo "i равно 1";
break;
case 2:
echo "i равно 2";
break;
}

// Эквивалентно:

if ($i == 0) {
echo "i равно 0";
} elseif ($i == 1) {
echo "i равно 1";
} elseif ($i == 2) {
echo "i равно 2";
}
?>
```

Пример #2 Оператор switch допускает сравнение с типом string

```
<?php
switch ($i) {
case "яблоко":
echo "i это яблоко";
break;
case "шоколадка":
echo "i это шоколадка";
break;
case "пирог":
echo "i это пирог";
break;
}
```

```
?>
```

Важно понять, как оператор `switch` выполняется, чтобы избежать ошибок. Оператор `switch` исполняет строчка за строчкой (на самом деле выражение за выражением). В начале никакой код не исполняется. Только в случае нахождения оператора `case`, значение которого совпадает со значением выражения в операторе `switch`, PHP начинает исполнять операторы. PHP продолжает исполнять операторы до конца блока `switch` либо до тех пор, пока не встретит оператор `break`. Если вы не напишете оператор `break` в конце секции `case`, PHP будет продолжать исполнять команды следующей секции `case`. Например :

```
<?php
switch ($i) {
case 0:
echo "i равно 0";
case 1:
echo "i равно 1";
case 2:
echo "i равно 2";
}
?>
```

В этом примере, если i равно 0, то PHP исполнит все операторы `echo`! Если i равно 1, PHP исполнит два последних оператора `echo`. Вы получите ожидаемое поведение оператора ('i равно 2' будет отображено) только, если i будет равно 2. Таким образом, важно не забывать об операторах `break` (даже если вы, возможно, хотите избежать его использования по назначению при определённых обстоятельствах).

В операторе `switch` выражение вычисляется один раз и этот результат сравнивается с каждым оператором `case`. В выражении `elseif`, выражение вычисляется снова. Если ваше условие более сложное, чем простое сравнение и/или находится в цикле, конструкция `switch` может работать быстрее.

Список операторов для исполнения в секции `case` также может быть пустым, что просто передаёт управление списку операторов в следующей секции `case`.

```
<?php
switch ($i) {
case 0:
case 1:
case 2:
echo "i меньше чем 3, но неотрицательный";
break;
case 3:
echo "i равно 3";
}
?>
```

Специальный вид конструкции `case - default`. Сюда управление попадает тогда, когда не сработал ни один из других операторов `case`. Например:

```
<?php
switch ($i) {
case 0:
echo "i равно 0";
break;
case 1:
echo "i равно 1";
break;
case 2:
echo "i равно 2";
break;
default:
echo "i не равно 0, 1 или 2";
}
?>
```

Замечание: Несколько указаний `default` вызовут ошибку `E_COMPILE_ERROR`.

Замечание: Формально конструкция `default` может быть перечислена в любом месте. Она будет использоваться только в том случае, если ни один другой вариант не подходит. Однако, по соглашению, лучше всего поместить её в конец, как последнюю конструкцию.

Если ни одна конструкция `case` не совпадает и нет конструкции `default`, то код не будет выполнен, как если бы ни одно утверждение `if` не было истинным.

Значение `case` может быть задано в виде выражения. Однако это выражение будет оценено само по себе, а затем слабо сопоставлено со значением `switch`. Это означает, что его нельзя использовать для сложных оценок значения `switch`.

Например:

```
<?php
$target = 1;
$start = 3;

switch ($target) {
case $start - 1:
print "A";
break;
case $start - 2:
print "B";
break;
case $start - 3:
print "C";
break;
case $start - 4:
print "D";
break;
}

// Выведет "B"
?>
```

Для более сложных сравнений в качестве значения `switch` может использоваться значение **true**. Или, как вариант, вместо `switch` использовать блоки `if-else`.

```
<?php
$offset = 1;
$start = 3;

switch (true) {
case $start - $offset === 1:
print "A";
break;
case $start - $offset === 2:
print "B";
break;
case $start - $offset === 3:
print "C";
break;
case $start - $offset === 4:
print "D";
break;
}

// Выведет "B"
?>
```

Возможен альтернативный синтаксис для управляющей структуры `switch`. Для более детальной информации, смотрите [Альтернативный синтаксис для управляющих структур](#).

```
<?php
switch ($i):
case 0:
```

```
echo "i равно 0";
break;
case 1:
echo "i равно 1";
break;
case 2:
echo "i равно 2";
break;
default:
echo "i не равно 0, 1 или 2";
endswitch;
?>
```

Возможно использование точки с запятой вместо двоеточия после оператора case. К примеру :

```
<?php
switch($beer)
{
case 'tuborg';
case 'carlsberg';
case 'stella';
case 'heineken';
echo 'Хороший выбор';
break;
default;
echo 'Пожалуйста, сделайте новый выбор...';
break;
}
?>
```

Смотрите также

- [match](#)

[+add a note](#)

User Contributed Notes 6 notes

[up](#)

[down](#)

287

[MaxTheDragon at home dot nl ¶](#)

11 years ago

This is listed in the documentation above, but it's a bit tucked away between the paragraphs. The difference between a series of if statements and the switch statement is that the expression you're comparing with, is evaluated only once in a switch statement. I think this fact needs a little bit more attention, so here's an example:

```
<?php
$a = 0;

if(++$a == 3) echo 3;
elseif(++$a == 2) echo 2;
elseif(++$a == 1) echo 1;
else echo "No match!";

// Outputs: 2

$a = 0;

switch(++$a) {
case 3: echo 3; break;
case 2: echo 2; break;
case 1: echo 1; break;
```

```
default: echo "No match!"; break;
}
```

```
// Outputs: 1
?>
```

It is therefore perfectly safe to do:

```
<?php
switch(winNobelPrizeStartingFromBirth()) {
case "peace": echo "You won the Nobel Peace Prize!"; break;
case "physics": echo "You won the Nobel Prize in Physics!"; break;
case "chemistry": echo "You won the Nobel Prize in Chemistry!"; break;
case "medicine": echo "You won the Nobel Prize in Medicine!"; break;
case "literature": echo "You won the Nobel Prize in Literature!"; break;
default: echo "You bought a rusty iron medal from a shady guy who insists it's a Nobel Prize..."; break;
}
?>
```

without having to worry about the function being re-evaluated for every case. There's no need to preemptively save the result in a variable either.

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

115

[septerrianin at mail dot ru ¶](#)

5 years ago

php 7.2.8.

The answer to the eternal question " what is faster?":

1 000 000 000 iterations.

```
<?php
$s = time();
for ($i = 0; $i < 1000000000; ++$i) {
    $x = $i%10;
    if ($x == 1) {
        $y = $x * 1;
    } elseif ($x == 2) {
        $y = $x * 2;
    } elseif ($x == 3) {
        $y = $x * 3;
    } elseif ($x == 4) {
        $y = $x * 4;
    } elseif ($x == 5) {
        $y = $x * 5;
    } elseif ($x == 6) {
        $y = $x * 6;
    } elseif ($x == 7) {
        $y = $x * 7;
    } elseif ($x == 8) {
        $y = $x * 8;
    } elseif ($x == 9) {
        $y = $x * 9;
    } else {
        $y = $x * 10;
    }
}
print("if: " . (time() - $s) . "sec\n");
```

```
$s = time();
for ($i = 0; $i < 1000000000; ++$i) {
    $x = $i%10;
    switch ($x) {
```

```

case 1:
$y = $x * 1;
break;
case 2:
$y = $x * 2;
break;
case 3:
$y = $x * 3;
break;
case 4:
$y = $x * 4;
break;
case 5:
$y = $x * 5;
break;
case 6:
$y = $x * 6;
break;
case 7:
$y = $x * 7;
break;
case 8:
$y = $x * 8;
break;
case 9:
$y = $x * 9;
break;
default:
$y = $x * 10;
}
}
print("switch: " . (time() - $s) . "sec\n");
?>

```

Results:
if: 69sec
switch: 42sec

[up](#)

[down](#)

76

[nospam at please dot com ¶](#)

23 years ago

Just a trick I have picked up:

If you need to evaluate several variables to find the first one with an actual value, TRUE for instance. You can do it this was.

There is probably a better way but it has worked out well for me.

```
switch (true) {
```

```
case (X != 1):
```

```
case (Y != 1):
```

```
default:
```

```
}
```

[up](#)

[down](#)

3

[me at czarpino dot com ¶](#)

1 year ago

Although noted elsewhere, still worth noting is how loose comparison in switch-case was also affected by the change in string to number comparison. Prior PHP8, strings were converted to int before comparison. The reverse is now true which can cause issues for logic that relied on this behavior.

```
<?php
function testSwitch($key) {
    switch ($key) {
        case 'non numeric string':
            echo $key . ' matches "non numeric string"';
            break;
    }
}

testSwitch(0); // pre-PHP8, returns '0 matches "non numeric string"'
?>
```

[up](#)

[down](#)

2
[j dot kane dot third at gmail dot com ¶](#)

1 year ago

The default case appears to always be evaluated last. If break is excluded from the default case, then the proceeding cases will be reevaluated. This behavior appears to be undocumented.

```
<?php

$kinds = ['moo', 'kind1', 'kind2'];

foreach ($kinds as $kind) {
    switch($kind)
    {
        default:
            // The kind wasn't valid, set it to the default
            $kind = 'kind1';
            var_dump('default');

        case 'kind1':
            var_dump('1');
            break;

        case 'kind2':
            var_dump('2');
            break;

        case 'kindn':
            var_dump('n-th');
            break;
    }

    echo "\n\n";
}
```

?>

[up](#)

[down](#)

-2
[GeorgNation ¶](#)

3 months ago

You can wrap up the case/break block with a curly braces:

```
$x = 2;

switch ($x)
```

```
{
case 2: {
echo '2 entrypoint';
break;
}

default: {
echo 'default entrypoint';
break;
}
}
```

[+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](#)

