Search

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 »
« Переменные извне PHP

- Руководство по PHP
- Справочник языка

Change language: Russian

Submit a Pull Request Report a Bug

Константы

Содержание

Константа - это идентификатор (имя) для простого значения. Как следует из названия, это значение не может измениться в ходе выполнения скрипта (кроме магических констант, которые на самом деле не являются константами). Константы чувствительны к регистру. По принятому соглашению, имена констант всегда пишутся в верхнем регистре.

> **Замечание**:
>
> До PHP 8.0.0 константы, определённые с помощью функции define(), могли быть нечувствительными к регистру.

Имя константы должно соответствовать тем же правилам именования, что и другие имена в PHP. Правильное имя начинается с буквы или символа подчёркивания, за которым следует любое количество букв, цифр и символов подчёркивания. Регулярное выражение для проверки правильности имени константы выглядит так: `^[a-zA-Z_\x80-\xff][a-zA-Z0-9_\x80-\xff]*$`

Возможно определить константы с помощью функции define() зарезервированными или даже некорректными именами, значения которых могут быть получены только с помощью constant(). Однако, делать это не рекомендуется.

Подсказка

Смотрите также Руководство по именованию.

Пример #1 Правильные и неправильные имена констант

```
<?php

// Правильные имена констант
define("FOO", "что-то");
define("FOO2", "что-то ещё");
define("FOO_BAR", "что-то большее");

// Неправильные имена констант
define("2FOO", "что-то");

// Это верное объявление, но лучше его не использовать:
// PHP однажды может зарегистрировать волшебную константу,
// которая нарушит работу скрипта
define("__FOO__", "что-то");

?>
```

> **Замечание**: Понятие "буквы" здесь - это символы a-z, A-Z, и другие символы с ASCII-кодами от 128 до 255 (0x80-0xff).

Как и superglobals, константы доступны из любой области видимости. Константы можно использовать из любого места скрипта независимо от области видимости. Подробную информацию об областях видимости можно найти здесь.

> **Замечание**: Начиная с PHP 7.1.0, константе класса можно объявлять видимость защищённая или закрытая, делая её доступной только в иерархической области видимости класса, в котором она определена.

User Contributed Notes 9 notes

wbcarts at juno dot com ¶

11 years ago

```
11/14/2016 - note updated by sobak
-----

CONSTANTS and PHP Class Definitions

Using "define('MY_VAR', 'default value')" INSIDE a class definition does not work as expected. You have to use the PHP
keyword 'const' and initialize it with a scalar value -- boolean, int, float, string (or array in PHP 5.6+) -- right away.

<?php

define('MIN_VALUE', '0.0'); // RIGHT - Works OUTSIDE of a class definition.
define('MAX_VALUE', '1.0'); // RIGHT - Works OUTSIDE of a class definition.

//const MIN_VALUE = 0.0; RIGHT - Works both INSIDE and OUTSIDE of a class definition.
//const MAX_VALUE = 1.0; RIGHT - Works both INSIDE and OUTSIDE of a class definition.

class Constants
{
//define('MIN_VALUE', '0.0'); WRONG - Works OUTSIDE of a class definition.
//define('MAX_VALUE', '1.0'); WRONG - Works OUTSIDE of a class definition.

const MIN_VALUE = 0.0; // RIGHT - Works INSIDE of a class definition.
const MAX_VALUE = 1.0; // RIGHT - Works INSIDE of a class definition.

public static function getMinValue()
{
return self::MIN_VALUE;
}

public static function getMaxValue()
{
return self::MAX_VALUE;
}
}

?>

#Example 1:
You can access these constants DIRECTLY like so:
* type the class name exactly.
* type two (2) colons.
* type the const name exactly.

#Example 2:
Because our class definition provides two (2) static functions, you can also access them like so:
* type the class name exactly.
* type two (2) colons.
* type the function name exactly (with the parentheses).

<?php

#Example 1:
$min = Constants::MIN_VALUE;
$max = Constants::MAX_VALUE;

#Example 2:
$min = Constants::getMinValue();
$max = Constants::getMaxValue();
```

```
?>
```

Once class constants are declared AND initialized, they cannot be set to different values -- that is why there are no setMinValue() and setMaxValue() functions in the class definition -- which means they are READ-ONLY and STATIC (shared by all instances of the class).

up
down
21

gried at NOSPAM dot nsys dot by ¶

8 years ago

Lets expand comment of 'storm' about usage of undefined constants. His claim that 'An undefined constant evaluates as true...' is wrong and right at same time. As said further in documentation ' If you use an undefined constant, PHP assumes that you mean the name of the constant itself, just as if you called it as a string...'. So yeah, undefined global constant when accessed directly will be resolved as string equal to name of sought constant (as thought PHP supposes that programmer had forgot apostrophes and autofixes it) and non-zero non-empty string converts to True.

There are two ways to prevent this:
1. always use function constant('CONST_NAME') to get constant value (BTW it also works for class constants - constant('CLASS_NAME::CONST_NAME'));
2. use only class constants (that are defined inside of class using keyword const) because they are not converted to string when not found but throw exception instead (Fatal error: Undefined class constant).

up
down
23

katana at katana-inc dot com ¶

21 years ago

Warning, constants used within the heredoc syntax (http://www.php.net/manual/en/language.types.string.php) are not interpreted!

Editor's Note: This is true. PHP has no way of recognizing the constant from any other string of characters within the heredoc block.

up
down
12

warwick dot jm dot barnes at gmail dot com ¶

3 years ago

The documentation says, "You can access constants anywhere in your script without regard to scope", but it's worth keeping in mind that a const declaration must appear in the source file before the place where it's used.

This doesn't work (using PHP 5.4):

```
<?php
foo();
const X = 1;
function foo() {
echo "Value of X: " . X;
}
?>
```

Result: "Value of X: X"

But this works:

```
<?php
const X = 1;
foo();
function foo() {
echo "Value of X: " . X;
}
?>
```

Result: "Value of X: 1"

This is potentially confusing because you can refer to a function that occurs later in your source file, but not a constant. Even though the const declaration is processed at compile time, it behaves a bit like it's being processed at run time.

up
down
16
Raheel Khan ¶
8 years ago
class constant are by default public in nature but they cannot be assigned visibility factor and in turn gives syntax error

```
<?php

class constants {

const MAX_VALUE = 10;
public const MIN_VALUE =1;

}

// This will work
echo constants::MAX_VALUE;

// This will return syntax error
echo constants::MIN_VALUE;
?>
```

up
down
22
ewspencer at industrex dot com ¶
20 years ago
I find using the concatenation operator helps disambiguate value assignments with constants. For example, setting constants in a global configuration file:

```
<?php
define('LOCATOR', "/locator");
define('CLASSES', LOCATOR."/code/classes");
define('FUNCTIONS', LOCATOR."/code/functions");
define('USERDIR', LOCATOR."/user");
?>
```

Later, I can use the same convention when invoking a constant's value for static constructs such as require() calls:

```
<?php
require_once(FUNCTIONS."/database.fnc");
require_once(FUNCTIONS."/randchar.fnc");
?>
```

as well as dynamic constructs, typical of value assignment to variables:

```
<?php
$userid = randchar(8,'anc','u');
$usermap = USERDIR."/".$userid.".png";
?>
```

The above convention works for me, and helps produce self-documenting code.

-- Erich

up
down
14
Andreas R. ¶
16 years ago
If you are looking for predefined constants like
* PHP_OS (to show the operating system, PHP was compiled for; php_uname('s') might be more suitable),

* DIRECTORY_SEPARATOR ("\\" on Win, '/' Linux,...)
* PATH_SEPARATOR (';' on Win, ':' on Linux,...)
they are buried in 'Predefined Constants' under 'List of Reserved Words' in the appendix:
http://www.php.net/manual/en/reserved.constants.php
while the latter two are also mentioned in 'Directory Functions'
http://www.php.net/manual/en/ref.dir.php
up
down
13
hafenator2000 at yahoo dot com ¶
18 years ago
PHP Modules also define constants. Make sure to avoid constant name collisions. There are two ways to do this that I can think of.
First: in your code make sure that the constant name is not already used. ex. <?php if (! defined("CONSTANT_NAME")) { Define("CONSTANT_NAME","Some Value"); } ?> This can get messy when you start thinking about collision handling, and the implications of this.
Second: Use some off prepend to all your constant names without exception ex. <?php Define("SITE_CONSTANT_NAME","Some Value"); ?>

Perhaps the developers or documentation maintainers could recommend a good prepend and ask module writers to avoid that prepend in modules.
up
down
12
storm ¶
18 years ago
An undefined constant evaluates as true when not used correctly. Say for example you had something like this:

```
settings.php
<?php
// Debug mode
define('DEBUG',false);
?>

test.php
<?php
include('settings.php');

if (DEBUG) {
// echo some sensitive data.
}
?>
```

If for some reason settings.php doesn't get included and the DEBUG constant is not set, PHP will STILL print the sensitive data. The solution is to evaluate it. Like so:

```
settings.php
<?php
// Debug mode
define('DEBUG',0);
?>

test.php
<?php
include('settings.php');

if (DEBUG == 1) {
// echo some sensitive data.
}
?>
```

Now it works correctly.

+ add a note

- Справочник языка
  - Основы синтаксиса
  - Типы
  - Переменные
  - Константы
  - Выражения
  - Операторы
  - Управляющие конструкции
  - Функции
  - Классы и объекты
  - Пространства имён
  - Перечисления
  - Ошибки
  - Исключения
  - Fibers
  - Генераторы
  - Атрибуты
  - Объяснение ссылок
  - Предопределённые переменные
  - Предопределённые исключения
  - Встроенные интерфейсы и классы
  - Предопределённые атрибуты
  - Контекстные опции и параметры
  - Поддерживаемые протоколы и обёртки

- My PHP.net
- Contact
- Other PHP.net sites
- Privacy policy