## Basics

Represented by a dollar sign followed by the name of the variable & case-sensitive.

Variable names follow the same rules as other labels in PHP. A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus: '[a-zA-Z\_\x7f-\xff][a-zA-Z0-9\_\x7f-\xff]\*'

**Note**: For our purposes here, a letter is a-z, A-Z, and the bytes from 127 through 255 (0x7f-0xff).

**Note**: $this is a special variable that can't be assigned.

<?php  
$var = 'Bob';  
$Var = 'Joe';  
echo "$var, $Var";      // outputs "Bob, Joe"  
  
$4site = 'not yet';     // invalid; starts with a number  
$\_4site = 'not yet';    // valid; starts with an underscore  
$täyte = 'mansikka';    // valid; 'ä' is (Extended) ASCII 228.  
?>

By default, variables are always assigned by value. That is to say, when you assign an expression to a variable, the entire value of the original expression is copied into the destination variable. This means, for instance, that after assigning one variable's value to another, changing one of those variables will have no effect on the other.

PHP also offers another way to assign values to variables: [assign by reference](http://php.net/manual/en/language.references.php). This means that the new variable simply references (in other words, "becomes an alias for" or "points to") the original variable. Changes to the new variable affect the original, and vice versa.

To assign by reference, simply prepend an ampersand (&) to the beginning of the variable which is being assigned (the source variable). For instance, the following code snippet outputs 'My name is Bob' twice:

<?php  
$foo = 'Bob';              // Assign the value 'Bob' to $foo  
$bar = &$foo;              // Reference $foo via $bar.  
$bar = "My name is $bar";  // Alter $bar...  
echo $bar;  
echo $foo;                 // $foo is altered too.  
?>

One important thing to note is that only named variables may be assigned by reference.

<?php  
$foo = 25;  
$bar = &$foo;      // This is a valid assignment.  
$bar = &(24 \* 7);  // Invalid; references an unnamed expression.  
  
function test()  
{  
   return 25;  
}  
  
$bar = &test();    // Invalid.  
?>

It is not necessary to initialize variables in PHP however it is a very good practice. Uninitialized variables have a default value of their type depending on the context in which they are used - booleans default to **FALSE**, integers and floats default to zero, strings (e.g. used in [echo](http://php.net/manual/en/function.echo.php)) are set as an empty string and arrays become to an empty array.

**Example #1 Default values of uninitialized variables**

<?php  
// Unset AND unreferenced (no use context) variable; outputs NULL  
var\_dump($unset\_var);  
  
// Boolean usage; outputs 'false' (See ternary operators for more on this syntax)  
echo($unset\_bool ? "true\n" : "false\n");  
  
// String usage; outputs 'string(3) "abc"'  
$unset\_str .= 'abc';  
var\_dump($unset\_str);  
  
// Integer usage; outputs 'int(25)'  
$unset\_int += 25; // 0 + 25 => 25  
var\_dump($unset\_int);  
  
// Float/double usage; outputs 'float(1.25)'  
$unset\_float += 1.25;  
var\_dump($unset\_float);  
  
// Array usage; outputs array(1) {  [3]=>  string(3) "def" }  
$unset\_arr[3] = "def"; // array() + array(3 => "def") => array(3 => "def")  
var\_dump($unset\_arr);  
  
// Object usage; creates new stdClass object (see http://www.php.net/manual/en/reserved.classes.php)  
// Outputs: object(stdClass)#1 (1) {  ["foo"]=>  string(3) "bar" }  
$unset\_obj->foo = 'bar';  
var\_dump($unset\_obj);  
?>

Relying on the default value of an uninitialized variable is problematic in the case of including one file into another which uses the same variable name. It is also a major [security risk](http://php.net/manual/en/security.globals.php) with [register\_globals](http://php.net/manual/en/ini.core.php#ini.register-globals) turned on. E\_NOTICE level error is issued in case of working with uninitialized variables, however not in the case of appending elements to the uninitialized array. [isset()](http://php.net/manual/en/function.isset.php) language construct can be used to detect if a variable has been already initialized.

# Variable handling Functions

* [boolval](http://php.net/manual/en/function.boolval.php) — Get the boolean value of a variable
* [debug\_zval\_dump](http://php.net/manual/en/function.debug-zval-dump.php) — Dumps a string representation of an internal zend value to output
* [doubleval](http://php.net/manual/en/function.doubleval.php) — Alias of floatval
* [empty](http://php.net/manual/en/function.empty.php) — Determine whether a variable is empty
* [floatval](http://php.net/manual/en/function.floatval.php) — Get float value of a variable
* [get\_defined\_vars](http://php.net/manual/en/function.get-defined-vars.php) — Returns an array of all defined variables
* [get\_resource\_type](http://php.net/manual/en/function.get-resource-type.php) — Returns the resource type
* [gettype](http://php.net/manual/en/function.gettype.php) — Get the type of a variable
* [import\_request\_variables](http://php.net/manual/en/function.import-request-variables.php) — Import GET/POST/Cookie variables into the global scope
* [intval](http://php.net/manual/en/function.intval.php) — Get the integer value of a variable
* [is\_array](http://php.net/manual/en/function.is-array.php) — Finds whether a variable is an array
* [is\_bool](http://php.net/manual/en/function.is-bool.php) — Finds out whether a variable is a boolean
* [is\_callable](http://php.net/manual/en/function.is-callable.php) — Verify that the contents of a variable can be called as a function
* [is\_double](http://php.net/manual/en/function.is-double.php) — Alias of is\_float
* [is\_float](http://php.net/manual/en/function.is-float.php) — Finds whether the type of a variable is float
* [is\_int](http://php.net/manual/en/function.is-int.php) — Find whether the type of a variable is integer
* [is\_integer](http://php.net/manual/en/function.is-integer.php) — Alias of is\_int
* [is\_long](http://php.net/manual/en/function.is-long.php) — Alias of is\_int
* [is\_null](http://php.net/manual/en/function.is-null.php) — Finds whether a variable is NULL
* [is\_numeric](http://php.net/manual/en/function.is-numeric.php) — Finds whether a variable is a number or a numeric string
* [is\_object](http://php.net/manual/en/function.is-object.php) — Finds whether a variable is an object
* [is\_real](http://php.net/manual/en/function.is-real.php) — Alias of is\_float
* [is\_resource](http://php.net/manual/en/function.is-resource.php) — Finds whether a variable is a resource
* [is\_scalar](http://php.net/manual/en/function.is-scalar.php) — Finds whether a variable is a scalar
* [is\_string](http://php.net/manual/en/function.is-string.php) — Find whether the type of a variable is string
* [isset](http://php.net/manual/en/function.isset.php) — Determine if a variable is set and is not NULL
* [print\_r](http://php.net/manual/en/function.print-r.php) — Prints human-readable information about a variable
* [serialize](http://php.net/manual/en/function.serialize.php) — Generates a storable representation of a value
* [settype](http://php.net/manual/en/function.settype.php) — Set the type of a variable
* [strval](http://php.net/manual/en/function.strval.php) — Get string value of a variable
* [unserialize](http://php.net/manual/en/function.unserialize.php) — Creates a PHP value from a stored representation
* [unset](http://php.net/manual/en/function.unset.php) — Unset a given variable
* [var\_dump](http://php.net/manual/en/function.var-dump.php) — Dumps information about a variable
* [var\_export](http://php.net/manual/en/function.var-export.php) — Outputs or returns a parsable string representation of a variable

## Predefined Variables

PHP provides a large number of predefined variables to any script which it runs. Many of these variables, however, cannot be fully documented as they are dependent upon which server is running, the version and setup of the server, and other factors. Some of these variables will not be available when PHP is run on the [command line](http://php.net/manual/en/features.commandline.php). For a listing of these variables, please see the section on [Reserved Predefined Variables](http://php.net/manual/en/reserved.variables.php).

**Warning**

In PHP 4.2.0 and later, the default value for the PHP directive [register\_globals](http://php.net/manual/en/ini.core.php#ini.register-globals) is off. This is a major change in PHP. Having register\_globals off affects the set of predefined variables available in the global scope. For example, to get DOCUMENT\_ROOT you'll use [*$\_SERVER['DOCUMENT\_ROOT']*](http://php.net/manual/en/reserved.variables.server.php) instead of $DOCUMENT\_ROOT, or [*$\_GET['id']*](http://php.net/manual/en/reserved.variables.get.php) from the URL http://www.example.com/test.php?id=3 instead of $id, or [*$\_ENV['HOME']*](http://php.net/manual/en/reserved.variables.environment.php) instead of $HOME.

For related information on this change, read the configuration entry for [register\_globals](http://php.net/manual/en/ini.core.php#ini.register-globals), the security chapter on [Using Register Globals](http://php.net/manual/en/security.globals.php) , as well as the PHP [» 4.1.0](http://www.php.net/releases/4_1_0.php) and [» 4.2.0](http://www.php.net/releases/4_2_0.php) Release Announcements.

Using the available PHP Reserved Predefined Variables, like the [superglobal arrays](http://php.net/manual/en/language.variables.superglobals.php), is preferred.

From version 4.1.0 onward, PHP provides an additional set of predefined arrays containing variables from the web server (if applicable), the environment, and user input. These new arrays are rather special in that they are automatically global--i.e., automatically available in every scope. For this reason, they are often known as "superglobals". (There is no mechanism in PHP for user-defined superglobals.) The superglobals are listed below; however, for a listing of their contents and further discussion on PHP predefined variables and their natures, please see the section [Reserved Predefined Variables](http://php.net/manual/en/reserved.variables.php). Also, you'll notice how the older predefined variables ($HTTP\_\*\_VARS) still exist. As of PHP 5.0.0, the long PHP [predefined variable](http://php.net/manual/en/language.variables.predefined.php) arrays may be disabled with the [register\_long\_arrays](http://php.net/manual/en/ini.core.php#ini.register-long-arrays) directive.

**Note**: **Variable variables**

Superglobals cannot be used as [variable variables](http://php.net/manual/en/language.variables.variable.php) inside functions or class methods.

**Note**:

Even though both the superglobal and HTTP\_\*\_VARS can exist at the same time; they are not identical, so modifying one will not change the other.

If certain variables in [variables\_order](http://php.net/manual/en/ini.core.php#ini.variables-order) are not set, their appropriate PHP predefined arrays are also left empty.

## Variable scope

The scope of a variable is the context within which it is defined. For the most part all PHP variables only have a single scope. This single scope spans included and required files as well. For example:

<?php  
$a = 1;  
include 'b.inc';  
?>

Here the $a variable will be available within the included b.inc script. However, within user-defined functions a local function scope is introduced. Any variable used inside a function is by default limited to the local function scope. For example:

<?php  
$a = 1; /\* global scope \*/   
  
function test()  
{   
    echo $a; /\* reference to local scope variable \*/   
}   
  
test();  
?>

This script will not produce any output because the echo statement refers to a local version of the $a variable, and it has not been assigned a value within this scope. You may notice that this is a little bit different from the C language in that global variables in C are automatically available to functions unless specifically overridden by a local definition. This can cause some problems in that people may inadvertently change a global variable. In PHP global variables must be declared global inside a function if they are going to be used in that function.

### The global keyword

First, an example use of global:

**Example #1 Using global**

<?php  
$a = 1;  
$b = 2;  
  
function Sum()  
{  
    global $a, $b;  
  
    $b = $a + $b;  
}   
  
Sum();  
echo $b;  
?>

The above script will output 3. By declaring $a and $b global within the function, all references to either variable will refer to the global version. There is no limit to the number of global variables that can be manipulated by a function.

A second way to access variables from the global scope is to use the special PHP-defined [*$GLOBALS*](http://php.net/manual/en/reserved.variables.globals.php) array. The previous example can be rewritten as:

**Example #2 Using** [***$GLOBALS***](http://php.net/manual/en/reserved.variables.globals.php) **instead of global**

<?php  
$a = 1;  
$b = 2;  
  
function Sum()  
{  
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];  
}   
  
Sum();  
echo $b;  
?>

The [*$GLOBALS*](http://php.net/manual/en/reserved.variables.globals.php) array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element. Notice how [*$GLOBALS*](http://php.net/manual/en/reserved.variables.globals.php) exists in any scope, this is because [*$GLOBALS*](http://php.net/manual/en/reserved.variables.globals.php) is a [superglobal](http://php.net/manual/en/language.variables.superglobals.php). Here's an example demonstrating the power of superglobals:

**Example #3 Example demonstrating superglobals and scope**

<?php  
function test\_global()  
{  
    // Most predefined variables aren't "super" and require   
    // 'global' to be available to the functions local scope.  
    global $HTTP\_POST\_VARS;  
      
    echo $HTTP\_POST\_VARS['name'];  
      
    // Superglobals are available in any scope and do   
    // not require 'global'. Superglobals are available   
    // as of PHP 4.1.0, and HTTP\_POST\_VARS is now  
    // deemed deprecated.  
    echo $\_POST['name'];  
}  
?>

**Note**:

Using global keyword outside a function is not an error. It can be used if the file is included from inside a function.

### Using static variables

Another important feature of variable scoping is the static variable. A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope. Consider the following example:

**Example #4 Example demonstrating need for static variables**

<?php  
function test()  
{  
    $a = 0;  
    echo $a;  
    $a++;  
}  
?>

This function is quite useless since every time it is called it sets $a to 0 and prints 0. The $a++ which increments the variable serves no purpose since as soon as the function exits the $a variable disappears. To make a useful counting function which will not lose track of the current count, the $a variable is declared static:

**Example #5 Example use of static variables**

<?php  
function test()  
{  
    static $a = 0;  
    echo $a;  
    $a++;  
}  
?>

Now, $a is initialized only in first call of function and every time the test() function is called it will print the value of $a and increment it.

Static variables also provide one way to deal with recursive functions. A recursive function is one which calls itself. Care must be taken when writing a recursive function because it is possible to make it recurse indefinitely. You must make sure you have an adequate way of terminating the recursion. The following simple function recursively counts to 10, using the static variable $count to know when to stop:

**Example #6 Static variables with recursive functions**

<?php  
function test()  
{  
    static $count = 0;  
  
    $count++;  
    echo $count;  
    if ($count < 10) {  
        test();  
    }  
    $count--;  
}  
?>

**Note**:

Static variables may be declared as seen in the examples above. Trying to assign values to these variables which are the result of expressions will cause a parse error.

**Example #7 Declaring static variables**

<?php  
function foo(){  
    static $int = 0;          // correct   
    static $int = 1+2;        // wrong  (as it is an expression)  
    static $int = sqrt(121);  // wrong  (as it is an expression too)  
  
    $int++;  
    echo $int;  
}  
?>

**Note**:

Static declarations are resolved in compile-time.

### References with global and static variables

The Zend Engine 1, driving PHP 4, implements the [static](http://php.net/manual/en/language.variables.scope.php#language.variables.scope.static) and [global](http://php.net/manual/en/language.variables.scope.php#language.variables.scope.global) modifier for variables in terms of [references](http://php.net/manual/en/language.references.php). For example, a true global variable imported inside a function scope with the global statement actually creates a reference to the global variable. This can lead to unexpected behaviour which the following example addresses:

<?php  
function test\_global\_ref() {  
    global $obj;  
    $obj = &new stdclass;  
}  
  
function test\_global\_noref() {  
    global $obj;  
    $obj = new stdclass;  
}  
  
test\_global\_ref();  
var\_dump($obj);  
test\_global\_noref();  
var\_dump($obj);  
?>

The above example will output:

NULL  
object(stdClass)(0) {  
}

A similar behaviour applies to the static statement. References are not stored statically:

<?php  
function &get\_instance\_ref() {  
    static $obj;  
  
    echo 'Static object: ';  
    var\_dump($obj);  
    if (!isset($obj)) {  
        // Assign a reference to the static variable  
        $obj = &new stdclass;  
    }  
    $obj->property++;  
    return $obj;  
}  
  
function &get\_instance\_noref() {  
    static $obj;  
  
    echo 'Static object: ';  
    var\_dump($obj);  
    if (!isset($obj)) {  
        // Assign the object to the static variable  
        $obj = new stdclass;  
    }  
    $obj->property++;  
    return $obj;  
}  
  
$obj1 = get\_instance\_ref();  
$still\_obj1 = get\_instance\_ref();  
echo "\n";  
$obj2 = get\_instance\_noref();  
$still\_obj2 = get\_instance\_noref();  
?>

The above example will output:

Static object: NULL  
Static object: NULL  
  
Static object: NULL  
Static object: object(stdClass)(1) {  
["property"]=>  
int(1)  
}

This example demonstrates that when assigning a reference to a static variable, it's not remembered when you call the &get\_instance\_ref() function a second time.

## Variable variables

Sometimes it is convenient to be able to have variable variable names. That is, a variable name which can be set and used dynamically. A normal variable is set with a statement such as:

<?php  
$a = 'hello';  
?>

A variable variable takes the value of a variable and treats that as the name of a variable. In the above example, hello, can be used as the name of a variable by using two dollar signs. i.e.

<?php  
$$a = 'world';  
?>

At this point two variables have been defined and stored in the PHP symbol tree: $a with contents "hello" and $hello with contents "world". Therefore, this statement:

<?php  
echo "$a ${$a}";  
?>

produces the exact same output as:

<?php  
echo "$a $hello";  
?>

i.e. they both produce: hello world.

In order to use variable variables with arrays, you have to resolve an ambiguity problem. That is, if you write $$a[1] then the parser needs to know if you meant to use $a[1] as a variable, or if you wanted $$a as the variable and then the [1] index from that variable. The syntax for resolving this ambiguity is: ${$a[1]} for the first case and ${$a}[1] for the second.

Class properties may also be accessed using variable property names. The variable property name will be resolved within the scope from which the call is made. For instance, if you have an expression such as $foo->$bar, then the local scope will be examined for $bar and its value will be used as the name of the property of $foo. This is also true if $bar is an array access.

Curly braces may also be used, to clearly delimit the property name. They are most useful when accessing values within a property that contains an array, when the property name is made of mulitple parts, or when the property name contains characters that are not otherwise valid (e.g. from [json\_decode()](http://php.net/manual/en/function.json-decode.php) or [SimpleXML](http://php.net/manual/en/book.simplexml.php)).

**Example #1 Variable property example**

<?php  
class foo {  
    var $bar = 'I am bar.';  
    var $arr = array('I am A.', 'I am B.', 'I am C.');  
    var $r   = 'I am r.';  
}  
  
$foo = new foo();  
$bar = 'bar';  
$baz = array('foo', 'bar', 'baz', 'quux');  
echo $foo->$bar . "\n";  
echo $foo->$baz[1] . "\n";  
  
$start = 'b';  
$end   = 'ar';  
echo $foo->{$start . $end} . "\n";  
  
$arr = 'arr';  
echo $foo->$arr[1] . "\n";  
echo $foo->{$arr}[1] . "\n";  
  
?>

The above example will output:

I am bar.  
I am bar.  
I am bar.  
I am r.  
I am B.

**Warning**

Please note that variable variables cannot be used with PHP's [Superglobal arrays](http://php.net/manual/en/language.variables.superglobals.php) within functions or class methods. The variable $this is also a special variable that cannot be referenced dynamically.

## Variables From External Sources [¶](http://php.net/manual/en/language.variables.external.php#language.variables.external)

### HTML Forms (GET and POST) [¶](http://php.net/manual/en/language.variables.external.php#language.variables.external.form)

When a form is submitted to a PHP script, the information from that form is automatically made available to the script. There are few ways to access this information, for example:

**Example #1 A simple HTML form**

<form action="foo.php" method="post">

Name: <input type="text" name="username" /><br />

Email: <input type="text" name="email" /><br />

<input type="submit" name="submit" value="Submit me!" />

</form>

As of PHP 5.4.0, there are only two ways to access data from your HTML forms. Currently availiable methods are listed below:

**Example #2 Accessing data from a simple POST HTML form**

<?php  
echo $\_POST['username'];  
echo $\_REQUEST['username'];  
?>

There were some other ways of accessing user input in old PHP versions. There are listed below. See changelog at the bottom of the page for more details.

**Example #3 Old methods of accessing user input**

<?php  
// WATCH OUT: these methods ARE NOT supported anymore.  
// Valid ones were described above.  
  
// Using import\_request\_variables() - this function has been removed in PHP 5.4.0  
   import\_request\_variables('p', 'p\_');  
   echo $p\_username;  
  
// These long predefined variables were removed in PHP 5.4.0  
   echo $HTTP\_POST\_VARS['username'];  
  
// Using register\_globals. This feature was removed in PHP 5.4.0  
   echo $username;  
?>

Using a GET form is similar except you'll use the appropriate GET predefined variable instead. GET also applies to the QUERY\_STRING (the information after the '?' in a URL). So, for example, http://www.example.com/test.php?id=3 contains GET data which is accessible with [*$\_GET['id']*](http://php.net/manual/en/reserved.variables.get.php). See also [*$\_REQUEST*](http://php.net/manual/en/reserved.variables.request.php).

**Note**:

Dots and spaces in variable names are converted to underscores. For example <input name="a.b" /> becomes $\_REQUEST["a\_b"].

PHP also understands arrays in the context of form variables (see the [related faq](http://php.net/manual/en/faq.html.php)). You may, for example, group related variables together, or use this feature to retrieve values from a multiple select input. For example, let's post a form to itself and upon submission display the data:

**Example #4 More complex form variables**

<?php  
if ($\_POST) {  
    echo '<pre>';  
    echo htmlspecialchars(print\_r($\_POST, true));  
    echo '</pre>';  
}  
?>  
<form action="" method="post">  
    Name:  <input type="text" name="personal[name]" /><br />  
    Email: <input type="text" name="personal[email]" /><br />  
    Beer: <br />  
    <select multiple name="beer[]">  
        <option value="warthog">Warthog</option>  
        <option value="guinness">Guinness</option>  
        <option value="stuttgarter">Stuttgarter Schwabenbräu</option>  
    </select><br />  
    <input type="submit" value="submit me!" />  
</form>

#### IMAGE SUBMIT variable names

When submitting a form, it is possible to use an image instead of the standard submit button with a tag like:

<input type="image" src="image.gif" name="sub" />

When the user clicks somewhere on the image, the accompanying form will be transmitted to the server with two additional variables, sub\_x and sub\_y. These contain the coordinates of the user click within the image. The experienced may note that the actual variable names sent by the browser contains a period rather than an underscore, but PHP converts the period to an underscore automatically.

### HTTP Cookies

PHP transparently supports HTTP cookies as defined by [» RFC 6265](http://www.faqs.org/rfcs/rfc6265). Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. You can set cookies using the [setcookie()](http://php.net/manual/en/function.setcookie.php) function. Cookies are part of the HTTP header, so the SetCookie function must be called before any output is sent to the browser. This is the same restriction as for the [header()](http://php.net/manual/en/function.header.php) function. Cookie data is then available in the appropriate cookie data arrays, such as [*$\_COOKIE*](http://php.net/manual/en/reserved.variables.cookies.php) as well as in [*$\_REQUEST*](http://php.net/manual/en/reserved.variables.request.php). See the [setcookie()](http://php.net/manual/en/function.setcookie.php) manual page for more details and examples.

If you wish to assign multiple values to a single cookie variable, you may assign it as an array. For example:

<?php  
  setcookie("MyCookie[foo]", 'Testing 1', time()+3600);  
  setcookie("MyCookie[bar]", 'Testing 2', time()+3600);  
?>

That will create two separate cookies although MyCookie will now be a single array in your script. If you want to set just one cookie with multiple values, consider using [serialize()](http://php.net/manual/en/function.serialize.php) or [explode()](http://php.net/manual/en/function.explode.php) on the value first.

Note that a cookie will replace a previous cookie by the same name in your browser unless the path or domain is different. So, for a shopping cart application you may want to keep a counter and pass this along. i.e.

**Example #5 A** [**setcookie()**](http://php.net/manual/en/function.setcookie.php) **example**

<?php  
if (isset($\_COOKIE['count'])) {  
    $count = $\_COOKIE['count'] + 1;  
} else {  
    $count = 1;  
}  
setcookie('count', $count, time()+3600);  
setcookie("Cart[$count]", $item, time()+3600);  
?>

### Dots in incoming variable names [¶](http://php.net/manual/en/language.variables.external.php#language.variables.external.dot-in-names)

Typically, PHP does not alter the names of variables when they are passed into a script. However, it should be noted that the dot (period, full stop) is not a valid character in a PHP variable name. For the reason, look at it:

<?php  
$varname.ext;  /\* invalid variable name \*/  
?>

Now, what the parser sees is a variable named $varname, followed by the string concatenation operator, followed by the barestring (i.e. unquoted string which doesn't match any known key or reserved words) 'ext'. Obviously, this doesn't have the intended result.

For this reason, it is important to note that PHP will automatically replace any dots in incoming variable names with underscores.

### Determining variable types

Because PHP determines the types of variables and converts them (generally) as needed, it is not always obvious what type a given variable is at any one time. PHP includes several functions which find out what type a variable is, such as: [gettype()](http://php.net/manual/en/function.gettype.php), [is\_array()](http://php.net/manual/en/function.is-array.php), [is\_float()](http://php.net/manual/en/function.is-float.php), [is\_int()](http://php.net/manual/en/function.is-int.php), [is\_object()](http://php.net/manual/en/function.is-object.php), and [is\_string()](http://php.net/manual/en/function.is-string.php). See also the chapter on [Types](http://php.net/manual/en/language.types.php).

### Changelog [¶](http://php.net/manual/en/language.variables.external.php#language.variables.external.changelog)

| **Version** | **Description** |
| --- | --- |
| 5.4.0 | [Register Globals](http://php.net/manual/en/security.globals.php), [Magic Quotes](http://php.net/manual/en/security.magicquotes.php) and [register\_long\_arrays](http://php.net/manual/en/ini.core.php#ini.register-long-arrays) has been removed |
| 5.3.0 | [Register Globals](http://php.net/manual/en/security.globals.php), [Magic Quotes](http://php.net/manual/en/security.magicquotes.php) and [register\_long\_arrays](http://php.net/manual/en/ini.core.php#ini.register-long-arrays) became deprecated |
| 4.2.0 | [register\_globals](http://php.net/manual/en/ini.core.php#ini.register-globals) directive defaults to off. |
| 4.1.0 | [Superglobal arrays](http://php.net/manual/en/language.variables.superglobals.php), like [*$\_POST*](http://php.net/manual/en/reserved.variables.post.php) and [*$\_GET*](http://php.net/manual/en/reserved.variables.get.php) became available |

# Constants

* [Syntax](http://php.net/manual/en/language.constants.syntax.php)
* [Magic constants](http://php.net/manual/en/language.constants.predefined.php)

A constant is an identifier (name) for a simple value. As the name suggests, that value cannot change during the execution of the script (except for [magic constants](http://php.net/manual/en/language.constants.predefined.php), which aren't actually constants). A constant is case-sensitive by default. By convention, constant identifiers are always uppercase.

The name of a constant follows the same rules as any label in PHP. A valid constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thusly: [a-zA-Z\_\x7f-\xff][a-zA-Z0-9\_\x7f-\xff]\*

**Example #1 Valid and invalid constant names**

<?php  
  
// Valid constant names  
define("FOO",     "something");  
define("FOO2",    "something else");  
define("FOO\_BAR", "something more");  
  
// Invalid constant names  
define("2FOO",    "something");  
  
// This is valid, but should be avoided:  
// PHP may one day provide a magical constant  
// that will break your script  
define("\_\_FOO\_\_", "something");   
  
?>

**Note**: For our purposes here, a letter is a-z, A-Z, and the ASCII characters from 127 through 255 (0x7f-0xff).

Like [superglobals](http://php.net/manual/en/language.variables.predefined.php), the scope of a constant is global. You can access constants anywhere in your script without regard to scope. For more information on scope, read the manual section on [variable scope](http://php.net/manual/en/language.variables.scope.php).

## Syntax

You can define a constant by using the [define()](http://php.net/manual/en/function.define.php)-function or by using the const keyword outside a class definition as of PHP 5.3.0. Once a constant is defined, it can never be changed or undefined.

Only scalar data ([boolean](http://php.net/manual/en/language.types.boolean.php), [integer](http://php.net/manual/en/language.types.integer.php), [float](http://php.net/manual/en/language.types.float.php) and [string](http://php.net/manual/en/language.types.string.php)) can be contained in constants prior to PHP 5.6. From PHP 5.6 onwards, it is also possible to define an [array](http://php.net/manual/en/language.types.array.php) constant. It is possible to define constants as a [resource](http://php.net/manual/en/language.types.resource.php), but it should be avoided, as it can cause unexpected results.

You can get the value of a constant by simply specifying its name. Unlike with variables, you should not prepend a constant with a $. You can also use the function [constant()](http://php.net/manual/en/function.constant.php) to read a constant's value if you wish to obtain the constant's name dynamically. Use [get\_defined\_constants()](http://php.net/manual/en/function.get-defined-constants.php) to get a list of all defined constants.

**Note**: Constants and (global) variables are in a different namespace. This implies that for example **TRUE** and $TRUE are generally different.

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](http://php.net/manual/en/language.types.string.php) (CONSTANT vs "CONSTANT"). An error of level [E\_NOTICE](http://php.net/manual/en/ref.errorfunc.php) will be issued when this happens. See also the manual entry on why [$foo[bar]](http://php.net/manual/en/language.types.array.php#language.types.array.foo-bar) is wrong (unless you first [define()](http://php.net/manual/en/function.define.php) bar as a constant). If you simply want to check if a constant is set, use the [defined()](http://php.net/manual/en/function.defined.php) function.

These are the differences between constants and variables:

* Constants do not have a dollar sign ($) before them;
* Constants may only be defined using the [define()](http://php.net/manual/en/function.define.php) function, not by simple assignment;
* Constants may be defined and accessed anywhere without regard to variable scoping rules;
* Constants may not be redefined or undefined once they have been set; and
* Constants may only evaluate to scalar values, or scalar or array values in PHP 5.6 and later.

**Example #1 Defining Constants**

<?php  
define("CONSTANT", "Hello world.");  
echo CONSTANT; // outputs "Hello world."  
echo Constant; // outputs "Constant" and issues a notice.  
?>

**Example #2 Defining Constants using the const keyword**

<?php  
// Works as of PHP 5.3.0  
const CONSTANT = 'Hello World';  
  
echo CONSTANT;  
  
// Works as of PHP 5.6.0  
const ANOTHER\_CONST = CONSTANT.'; Goodbye World';  
  
echo ANOTHER\_CONST;  
?>

**Note**:

As opposed to defining constants using [define()](http://php.net/manual/en/function.define.php), constants defined using the const keyword must be declared at the top-level scope because they are defined at compile-time. This means that they cannot be declared inside functions, loops, if statements or try/ catch blocks.

See also [Class Constants](http://php.net/manual/en/language.oop5.constants.php)

## Magic constants

PHP provides a large number of [predefined constants](http://php.net/manual/en/reserved.constants.php) to any script which it runs. Many of these constants, however, are created by various extensions, and will only be present when those extensions are available, either via dynamic loading or because they have been compiled in.

There are eight magical constants that change depending on where they are used. For example, the value of **\_\_LINE\_\_** depends on the line that it's used on in your script. These special constants are case-insensitive and are as follows:

| **A few "magical" PHP constants** | |
| --- | --- |
| **Name** | **Description** |
| **\_\_LINE\_\_** | The current line number of the file. |
| **\_\_FILE\_\_** | The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, **\_\_FILE\_\_** always contains an absolute path with symlinks resolved whereas in older versions it contained relative path under some circumstances. |
| **\_\_DIR\_\_** | The directory of the file. If used inside an include, the directory of the included file is returned. This is equivalent to dirname(\_\_FILE\_\_). This directory name does not have a trailing slash unless it is the root directory. (Added in PHP 5.3.0.) |
| **\_\_FUNCTION\_\_** | The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased. |
| **\_\_CLASS\_\_** | The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased. The class name includes the namespace it was declared in (e.g. Foo\Bar). Note that as of PHP 5.4 \_\_CLASS\_\_ works also in traits. When used in a trait method, \_\_CLASS\_\_ is the name of the class the trait is used in. |
| **\_\_TRAIT\_\_** | The trait name. (Added in PHP 5.4.0) As of PHP 5.4 this constant returns the trait as it was declared (case-sensitive). The trait name includes the namespace it was declared in (e.g. Foo\Bar). |
| **\_\_METHOD\_\_** | The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive). |
| **\_\_NAMESPACE\_\_** | The name of the current namespace (case-sensitive). This constant is defined in compile-time (Added in PHP 5.3.0). |

See also [get\_class()](http://php.net/manual/en/function.get-class.php), [get\_object\_vars()](http://php.net/manual/en/function.get-object-vars.php), [file\_exists()](http://php.net/manual/en/function.file-exists.php) and [function\_exists()](http://php.net/manual/en/function.function-exists.php).

## Predefined Constants

### Core Predefined Constants

These constants are defined by the PHP core. This includes PHP, the Zend engine, and SAPI modules.

**PHP\_VERSION** ([string](http://php.net/manual/en/language.types.string.php))

The current PHP version as a string in "major.minor.release[extra]" notation.

**PHP\_MAJOR\_VERSION** ([integer](http://php.net/manual/en/language.types.integer.php))

The current PHP "major" version as an integer (e.g., int(5) from version "5.2.7-extra"). Available since PHP 5.2.7.

**PHP\_MINOR\_VERSION** ([integer](http://php.net/manual/en/language.types.integer.php))

The current PHP "minor" version as an integer (e.g., int(2) from version "5.2.7-extra"). Available since PHP 5.2.7.

**PHP\_RELEASE\_VERSION** ([integer](http://php.net/manual/en/language.types.integer.php))

The current PHP "release" version as an integer (e.g., int(7) from version "5.2.7-extra"). Available since PHP 5.2.7.

**PHP\_VERSION\_ID** ([integer](http://php.net/manual/en/language.types.integer.php))

The current PHP version as an integer, useful for version comparisons (e.g., int(50207) from version "5.2.7-extra"). Available since PHP 5.2.7.

**PHP\_EXTRA\_VERSION** ([string](http://php.net/manual/en/language.types.string.php))

The current PHP "extra" version as a string (e.g., '-extra' from version "5.2.7-extra"). Often used by distribution vendors to indicate a package version. Available since PHP 5.2.7.

**PHP\_ZTS** ([integer](http://php.net/manual/en/language.types.integer.php))

Available since PHP 5.2.7.

**PHP\_DEBUG** ([integer](http://php.net/manual/en/language.types.integer.php))

Available since PHP 5.2.7.

**PHP\_MAXPATHLEN** ([integer](http://php.net/manual/en/language.types.integer.php))

The maximum length of filenames (including path) supported by this build of PHP. Available since PHP 5.3.0.

**PHP\_OS** ([string](http://php.net/manual/en/language.types.string.php))

**PHP\_SAPI** ([string](http://php.net/manual/en/language.types.string.php))

The Server API for this build of PHP. Available since PHP 4.2.0. See also [php\_sapi\_name()](http://php.net/manual/en/function.php-sapi-name.php).

**PHP\_EOL** ([string](http://php.net/manual/en/language.types.string.php))

The correct 'End Of Line' symbol for this platform. Available since PHP 4.3.10 and PHP 5.0.2

**PHP\_INT\_MAX** ([integer](http://php.net/manual/en/language.types.integer.php))

The largest integer supported in this build of PHP. Usually int(2147483647). Available since PHP 4.4.0 and PHP 5.0.5

**PHP\_INT\_SIZE** ([integer](http://php.net/manual/en/language.types.integer.php))

Available since PHP 4.4.0 and PHP 5.0.5

**DEFAULT\_INCLUDE\_PATH** ([string](http://php.net/manual/en/language.types.string.php))

**PEAR\_INSTALL\_DIR** ([string](http://php.net/manual/en/language.types.string.php))

**PEAR\_EXTENSION\_DIR** ([string](http://php.net/manual/en/language.types.string.php))

**PHP\_EXTENSION\_DIR** ([string](http://php.net/manual/en/language.types.string.php))

**PHP\_PREFIX** ([string](http://php.net/manual/en/language.types.string.php))

The value "--prefix" was set to at configure. Available since PHP 4.3.0.

**PHP\_BINDIR** ([string](http://php.net/manual/en/language.types.string.php))

Specifies where the binaries were installed into.

**PHP\_BINARY** ([string](http://php.net/manual/en/language.types.string.php))

Specifies the PHP binary path during script execution. Available since PHP 5.4.

**PHP\_MANDIR** ([string](http://php.net/manual/en/language.types.string.php))

Specifies where the manpages were installed into. Available since PHP 5.3.7.

**PHP\_LIBDIR** ([string](http://php.net/manual/en/language.types.string.php))

**PHP\_DATADIR** ([string](http://php.net/manual/en/language.types.string.php))

**PHP\_SYSCONFDIR** ([string](http://php.net/manual/en/language.types.string.php))

**PHP\_LOCALSTATEDIR** ([string](http://php.net/manual/en/language.types.string.php))

**PHP\_CONFIG\_FILE\_PATH** ([string](http://php.net/manual/en/language.types.string.php))

**PHP\_CONFIG\_FILE\_SCAN\_DIR** ([string](http://php.net/manual/en/language.types.string.php))

**PHP\_SHLIB\_SUFFIX** ([string](http://php.net/manual/en/language.types.string.php))

The build-platform's shared library suffix, such as "so" (most Unixes) or "dll" (Windows). Available since PHP 4.3.0

**E\_ERROR** ([integer](http://php.net/manual/en/language.types.integer.php))

[Error reporting constant](http://php.net/manual/en/errorfunc.constants.php)

**E\_WARNING** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_PARSE** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_NOTICE** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_CORE\_ERROR** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_CORE\_WARNING** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_COMPILE\_ERROR** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_COMPILE\_WARNING** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_USER\_ERROR** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_USER\_WARNING** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_USER\_NOTICE** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_DEPRECATED** ([integer](http://php.net/manual/en/language.types.integer.php))

Available since PHP 5.3.0

**E\_USER\_DEPRECATED** ([integer](http://php.net/manual/en/language.types.integer.php))

Available since PHP 5.3.0

**E\_ALL** ([integer](http://php.net/manual/en/language.types.integer.php))

**E\_STRICT** ([integer](http://php.net/manual/en/language.types.integer.php))

Available since PHP 5.0.0

**\_\_COMPILER\_HALT\_OFFSET\_\_** ([integer](http://php.net/manual/en/language.types.integer.php))

Available since PHP 5.1.0

**TRUE** ([boolean](http://php.net/manual/en/language.types.boolean.php))

See [Booleans](http://php.net/manual/en/language.types.boolean.php).

**FALSE** ([boolean](http://php.net/manual/en/language.types.boolean.php))

See [Booleans](http://php.net/manual/en/language.types.boolean.php).

**NULL** ([null](http://php.net/manual/en/language.types.null.php))

See [Null](http://php.net/manual/en/language.types.null.php).

See also: [Magic constants](http://php.net/manual/en/language.constants.predefined.php).

# Expressions

Expressions are the most important building stones of PHP. In PHP, almost anything you write is an expression. The simplest yet most accurate way to define an expression is "anything that has a value".

The most basic forms of expressions are constants and variables. When you type "$a = 5", you're assigning '5' into $a. '5', obviously, has the value 5, or in other words '5' is an expression with the value of 5 (in this case, '5' is an integer constant).

After this assignment, you'd expect $a's value to be 5 as well, so if you wrote $b = $a, you'd expect it to behave just as if you wrote $b = 5. In other words, $a is an expression with the value of 5 as well. If everything works right, this is exactly what will happen.

Slightly more complex examples for expressions are functions. For instance, consider the following function:

<?php  
function foo ()  
{  
    return 5;  
}  
?>

Assuming you're familiar with the concept of functions (if you're not, take a look at the chapter about [functions](http://php.net/manual/en/language.functions.php)), you'd assume that typing $c = foo() is essentially just like writing $c = 5, and you're right. Functions are expressions with the value of their return value. Since foo() returns 5, the value of the expression 'foo()' is 5. Usually functions don't just return a static value but compute something.

Of course, values in PHP don't have to be integers, and very often they aren't. PHP supports four scalar value types: [integer](http://php.net/manual/en/language.types.integer.php) values, floating point values ([float](http://php.net/manual/en/language.types.float.php)), [string](http://php.net/manual/en/language.types.string.php) values and [boolean](http://php.net/manual/en/language.types.boolean.php) values (scalar values are values that you can't 'break' into smaller pieces, unlike arrays, for instance). PHP also supports two composite (non-scalar) types: arrays and objects. Each of these value types can be assigned into variables or returned from functions.

PHP takes expressions much further, in the same way many other languages do. PHP is an expression-oriented language, in the sense that almost everything is an expression. Consider the example we've already dealt with, '$a = 5'. It's easy to see that there are two values involved here, the value of the integer constant '5', and the value of $a which is being updated to 5 as well. But the truth is that there's one additional value involved here, and that's the value of the assignment itself. The assignment itself evaluates to the assigned value, in this case 5. In practice, it means that '$a = 5', regardless of what it does, is an expression with the value 5. Thus, writing something like '$b = ($a = 5)' is like writing '$a = 5; $b = 5;' (a semicolon marks the end of a statement). Since assignments are parsed in a right to left order, you can also write '$b = $a = 5'.

Another good example of expression orientation is pre- and post-increment and decrement. Users of PHP and many other languages may be familiar with the notation of variable++ and variable--. These are [increment and decrement operators](http://php.net/manual/en/language.operators.increment.php). In PHP, like in C, there are two types of increment - pre-increment and post-increment. Both pre-increment and post-increment essentially increment the variable, and the effect on the variable is identical. The difference is with the value of the increment expression. Pre-increment, which is written '++$variable', evaluates to the incremented value (PHP increments the variable before reading its value, thus the name 'pre-increment'). Post-increment, which is written '$variable++' evaluates to the original value of $variable, before it was incremented (PHP increments the variable after reading its value, thus the name 'post-increment').

A very common type of expressions are [comparison](http://php.net/manual/en/language.operators.comparison.php) expressions. These expressions evaluate to either **FALSE** or **TRUE**. PHP supports > (bigger than), >= (bigger than or equal to), == (equal), != (not equal), < (smaller than) and <= (smaller than or equal to). The language also supports a set of strict equivalence operators: === (equal to and same type) and !== (not equal to or not same type). These expressions are most commonly used inside conditional execution, such as if statements.

The last example of expressions we'll deal with here is combined operator-assignment expressions. You already know that if you want to increment $a by 1, you can simply write '$a++' or '++$a'. But what if you want to add more than one to it, for instance 3? You could write '$a++' multiple times, but this is obviously not a very efficient or comfortable way. A much more common practice is to write '$a = $a + 3'. '$a + 3' evaluates to the value of $a plus 3, and is assigned back into $a, which results in incrementing $a by 3. In PHP, as in several other languages like C, you can write this in a shorter way, which with time would become clearer and quicker to understand as well. Adding 3 to the current value of $a can be written '$a += 3'. This means exactly "take the value of $a, add 3 to it, and assign it back into $a". In addition to being shorter and clearer, this also results in faster execution. The value of '$a += 3', like the value of a regular assignment, is the assigned value. Notice that it is NOT 3, but the combined value of $a plus 3 (this is the value that's assigned into $a). Any two-place operator can be used in this operator-assignment mode, for example '$a -= 5' (subtract 5 from the value of $a), '$b \*= 7' (multiply the value of $b by 7), etc.

There is one more expression that may seem odd if you haven't seen it in other languages, the ternary conditional operator:

<?php  
$first ? $second : $third  
?>

If the value of the first subexpression is **TRUE** (non-zero), then the second subexpression is evaluated, and that is the result of the conditional expression. Otherwise, the third subexpression is evaluated, and that is the value.

The following example should help you understand pre- and post-increment and expressions in general a bit better:

<?php  
function double($i)  
{  
    return $i\*2;  
}  
$b = $a = 5;        /\* assign the value five into the variable $a and $b \*/  
$c = $a++;          /\* post-increment, assign original value of $a   
                       (5) to $c \*/  
$e = $d = ++$b;     /\* pre-increment, assign the incremented value of   
                       $b (6) to $d and $e \*/  
  
/\* at this point, both $d and $e are equal to 6 \*/  
  
$f = double($d++);  /\* assign twice the value of $d before  
                       the increment, 2\*6 = 12 to $f \*/  
$g = double(++$e);  /\* assign twice the value of $e after  
                       the increment, 2\*7 = 14 to $g \*/  
$h = $g += 10;      /\* first, $g is incremented by 10 and ends with the   
                       value of 24. the value of the assignment (24) is   
                       then assigned into $h, and $h ends with the value   
                       of 24 as well. \*/  
?>

Some expressions can be considered as statements. In this case, a statement has the form of 'expr ;' that is, an expression followed by a semicolon. In '$b = $a = 5;', '$a = 5' is a valid expression, but it's not a statement by itself. '$b = $a = 5;' however is a valid statement.

One last thing worth mentioning is the truth value of expressions. In many events, mainly in conditional execution and loops, you're not interested in the specific value of the expression, but only care about whether it means **TRUE** or **FALSE**. The constants **TRUE** and **FALSE** (case-insensitive) are the two possible boolean values. When necessary, an expression is automatically converted to boolean. See the [section about type-casting](http://php.net/manual/en/language.types.type-juggling.php#language.types.typecasting) for details about how.

PHP provides a full and powerful implementation of expressions, and documenting it entirely goes beyond the scope of this manual. The above examples should give you a good idea about what expressions are and how you can construct useful expressions. Throughout the rest of this manual we'll write expr to indicate any valid PHP expression.

# 

**Next Topics**

## Operators

## Control Structures

## Language Constructs and Functions

## Namespaces

## Extensions

## Config

## Performance/bytecode caching \*

As an addendum to David's 10-Nov-2005 posting, remember that curly braces literally mean "evaluate what's inside the curly braces" so, you can squeeze the variable variable creation into one line, like this:  
  
<?php  
  ${"title\_default\_" . $title} = "selected";  
?>  
  
and then, for example:  
  
<?php  
  $title\_select = <<<END  
    <select name="title">  
      <option>Select</option>  
      <option $title\_default\_Mr  value="Mr">Mr</option>  
      <option $title\_default\_Ms  value="Ms">Ms</option>  
      <option $title\_default\_Mrs value="Mrs">Mrs</option>  
      <option $title\_default\_Dr  value="Dr">Dr</option>  
    </select>  
END;  
?>

[up](http://php.net/manual/vote-note.php?id=58632&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=58632&page=language.variables&vote=down)

-4

[***david at removethisbit dot futuresbright dot com*** ¶](http://php.net/manual/en/language.variables.php#58632)

**8 years ago**

When using variable variables this is invalid:   
  
<?php   
$my\_variable\_{$type}\_name = true;   
?>   
  
to get around this do something like:   
  
<?php   
$n="my\_variable\_{$type}\_name";   
${$n} = true;   
?>   
  
(or $$n - I tend to use curly brackets out of habit as it helps t reduce bugs ...)

[up](http://php.net/manual/vote-note.php?id=56351&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=56351&page=language.variables&vote=down)

-4

[***Chris Hester*** ¶](http://php.net/manual/en/language.variables.php#56351)

**8 years ago**

Variables can also be assigned together.  
  
<?php  
$a = $b = $c = 1;  
echo $a.$b.$c;  
?>  
  
This outputs 111.

[up](http://php.net/manual/vote-note.php?id=84581&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=84581&page=language.variables&vote=down)

-7

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.php#84581)

**5 years ago**

[EDIT by danbrown AT php DOT net: The function provided by this author will give you all defined variables at runtime.  It was originally written by (john DOT t DOT gold AT gmail DOT com), but contained some errors that were corrected in subsequent posts by (ned AT wgtech DOT com) and (taliesin AT gmail DOT com).]   
  
<?php   
  
echo '<table border=1><tr> <th>variable</th> <th>value</th> </tr>';   
foreach( get\_defined\_vars() as $key => $value)   
{   
    if (is\_array ($value) )   
    {   
        echo '<tr><td>$'.$key .'</td><td>';   
        if ( sizeof($value)>0 )   
        {   
        echo '"<table border=1><tr> <th>key</th> <th>value</th> </tr>';   
        foreach ($value as $skey => $svalue)   
        {   
            echo '<tr><td>[' . $skey .']</td><td>"'. $svalue .'"</td></tr>';   
        }   
        echo '</table>"';   
        }   
             else   
        {   
            echo 'EMPTY';   
        }   
        echo '</td></tr>';   
    }   
    else   
    {   
            echo '<tr><td>$' . $key .'</td><td>"'. $value .'"</td></tr>';   
    }   
}   
echo '</table>';   
?>

[up](http://php.net/manual/vote-note.php?id=39194&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=39194&page=language.variables&vote=down)

-7

[***webmaster at daersys dot net*** ¶](http://php.net/manual/en/language.variables.php#39194)

**10 years ago**

You don't necessarily have to escape the dollar-sign before a variable if you want to output its name.   
  
You can use single quotes instead of double quotes, too.   
  
For instance:   
  
<?php   
$var = "test";   
  
echo "$var"; // Will output the string "test"   
  
echo "\$var"; // Will output the string "$var"   
  
echo '$var'; // Will do the exact same thing as the previous line   
?>   
  
Why?   
Well, the reason for this is that the PHP Parser will not attempt to parse strings encapsulated in single quotes (as opposed to strings within double quotes) and therefore outputs exactly what it's being fed with :)   
  
To output the value of a variable within a single-quote-encapsulated string you'll have to use something along the lines of the following code:   
  
<?php   
$var = 'test';   
/\*   
Using single quotes here seeing as I don't need the parser to actually parse the content of this variable but merely treat it as an ordinary string   
\*/   
  
echo '$var = "' . $var . '"';   
/\*   
Will output:   
$var = "test"   
\*/   
?>   
  
HTH   
- Daerion

[up](http://php.net/manual/vote-note.php?id=42653&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=42653&page=language.variables&vote=down)

-10

[***raja shahed at christine nothdurfter dot com*** ¶](http://php.net/manual/en/language.variables.php#42653)

**10 years ago**

<?php  
error\_reporting(E\_ALL);  
  
$name = "Christine\_Nothdurfter";  
// not Christine Nothdurfter  
// you are not allowed to leave a space inside a variable name ;)  
$$name = "'s students of Tyrolean language ";  
  
print " $name{$$name}<br>";  
print  "$name$Christine\_Nothdurfter";  
// same  
?>

[up](http://php.net/manual/vote-note.php?id=59088&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=59088&page=language.variables&vote=down)

-9

[***Mike at ImmortalSoFar dot com*** ¶](http://php.net/manual/en/language.variables.php#59088)

**8 years ago**

References and "return" can be flakey:  
  
<?php  
//  This only returns a copy, despite the dereferencing in the function definition  
function &GetLogin ()  
{  
    return $\_SESSION['Login'];  
}  
  
//  This gives a syntax error  
function &GetLogin ()  
{  
    return &$\_SESSION['Login'];  
}  
  
//  This works  
function &GetLogin ()  
{  
    $ret = &$\_SESSION['Login'];  
    return $ret;  
}  
?>

[up](http://php.net/manual/vote-note.php?id=52924&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=52924&page=language.variables&vote=down)

-10

[***josh at PraxisStudios dot com*** ¶](http://php.net/manual/en/language.variables.php#52924)

**9 years ago**

As with echo, you can define a variable like this:   
  
<?php   
  
$text = <<<END   
  
<table>   
    <tr>   
        <td>   
             $outputdata   
        </td>   
     </tr>   
</table>   
  
END;   
  
?>   
  
The closing END; must be on a line by itself (no whitespace).   
  
[EDIT by danbrown AT php DOT net: This note illustrates HEREDOC syntax.  For more information on this and similar features, please read the "Strings" section of the manual here: <http://www.php.net/manual/en/language.types.string.php> ]

[up](http://php.net/manual/vote-note.php?id=48798&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=48798&page=language.variables&vote=down)

-11

[***Carel Solomon*** ¶](http://php.net/manual/en/language.variables.php#48798)

**9 years ago**

You can also construct a variable name by concatenating two different variables, such as:   
  
<?php   
  
$arg = "foo";   
$val = "bar";   
  
//${$arg$val} = "in valid";     // Invalid   
${$arg . $val} = "working";   
  
echo $foobar;     // "working";   
//echo $arg$val;         // Invalid   
//echo ${$arg$val};     // Invalid   
echo ${$arg . $val};    // "working"   
  
?>   
  
Carel

[up](http://php.net/manual/vote-note.php?id=99132&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=99132&page=language.variables&vote=down)

-13

[***dimitrov dot adrian at gmail dot com*** ¶](http://php.net/manual/en/language.variables.php#99132)

**3 years ago**

This is mine type casting lib, that is very useful for me.  
  
<?php   
  
function CAST\_TO\_INT($var, $min = FALSE, $max = FALSE)  
{  
    $var = is\_int($var) ? $var : (int)(is\_scalar($var) ? $var : 0);  
    if ($min !== FALSE && $var < $min)  
        return $min;  
    elseif($max !== FALSE && $var > $max)  
        return $max;  
    return $var;  
          
}  
  
function CAST\_TO\_FLOAT($var, $min = FALSE, $max = FALSE)  
{  
    $var = is\_float($var) ? $var : (float)(is\_scalar($var) ? $var : 0);  
    if ($min !== FALSE && $var < $min)  
        return $min;  
    elseif($max !== FALSE && $var > $max)  
        return $max;  
    return $var;  
}  
  
function CAST\_TO\_BOOL($var)  
{  
    return (bool)(is\_bool($var) ? $var : is\_scalar($var) ? $var : FALSE);  
}  
  
function CAST\_TO\_STRING($var, $length = FALSE)  
{  
    if ($length !== FALSE && is\_int($length) && $length > 0)  
        return substr(trim(is\_string($var)  
                    ? $var  
                    : (is\_scalar($var) ? $var : '')), 0, $length);  
  
    return trim(  
                is\_string($var)  
                ? $var  
                : (is\_scalar($var) ? $var : ''));  
}  
  
function CAST\_TO\_ARRAY($var)  
{  
    return is\_array($var)  
            ? $var  
            : is\_scalar($var) && $var  
                ? array($var)  
                : is\_object($var) ? (array)$var : array();  
}  
  
function CAST\_TO\_OBJECT($var)  
{  
    return is\_object($var)  
            ? $var  
            : is\_scalar($var) && $var  
                ? (object)$var  
                : is\_array($var) ? (object)$var : (object)NULL;  
}  
  
?>

[up](http://php.net/manual/vote-note.php?id=99211&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=99211&page=language.variables&vote=down)

-14

[***justgook at gmail dot com*** ¶](http://php.net/manual/en/language.variables.php#99211)

**3 years ago**

I found interstate solution to work with arrays  
  
<?php  
$vars['product']['price']=11;  
  
$aa='product';  
$bb='price';  
  
echo $vars{$aa}{$bb};  
  
//prints 11  
?>

[up](http://php.net/manual/vote-note.php?id=114124&page=language.variables&vote=up)

[down](http://php.net/manual/vote-note.php?id=114124&page=language.variables&vote=down)

-14

[***anoniempje*** ¶](http://php.net/manual/en/language.variables.php#114124)

**5 months ago**

<?php  
  function tafel\_vier()  
  {  
    $getal = 4;  
    for($i=1; $i <= 10; $i++)  
    {  
      $uitkomst = $i \* $getal;  
      echo $i . " maal " . $getal . " = " . $uitkomst . "<br/>";  
    }  
  }  
  tafel\_vier();  
?>

This page should include a note on variable lifecycle:  
  
Before a variable is used, it has no existence. It is unset. It is possible to check if a variable doesn't exist by using isset(). This returns true provided the variable exists and isn't set to null. With the exception of null, the value a variable holds plays no part in determining whether a variable is set.   
  
Setting an existing variable to null is a way of unsetting a variable. Another way is variables may be destroyed by using the unset() construct.   
  
<?php  
print isset($a); // $a is not set. Prints false. (Or more accurately prints ''.)  
$b = 0; // isset($b) returns true (or more accurately '1')  
$c = array(); // isset($c) returns true  
$b = null; // Now isset($b) returns false;  
unset($c); // Now isset($c) returns false;  
?>  
  
is\_null() is an equivalent test to checking that isset() is false.  
  
The first time that a variable is used in a scope, it's automatically created. After this isset is true. At the point at which it is created it also receives a type according to the context.  
  
<?php  
$a\_bool = true;   // a boolean  
$a\_str = 'foo';    // a string  
?>  
  
If it is used without having been given a value then it is uninitalized and it receives the default value for the type. The default values are the \_empty\_ values. E.g  Booleans default to FALSE, integers and floats default to zero, strings to the empty string '', arrays to the empty array.  
  
A variable can be tested for emptiness using empty();  
  
<?php  
$a = 0; //This isset, but is empty  
?>  
  
Unset variables are also empty.  
  
<?php  
empty($vessel); // returns true. Also $vessel is unset.  
?>  
  
Everything above applies to array elements too.   
  
<?php  
$item = array();   
//Now isset($item) returns true. But isset($item['unicorn']) is false.  
//empty($item) is true, and so is empty($item['unicorn']  
  
$item['unicorn'] = '';  
//Now isset($item['unicorn']) is true. And empty($item) is false.   
//But empty($item['unicorn']) is still true;  
  
$item['unicorn'] = 'Pink unicorn';  
//isset($item['unicorn']) is still true. And empty($item) is still false.   
//But now empty($item['unicorn']) is false;  
?>  
  
For arrays, this is important because accessing a non-existent array item can trigger errors; you may want to test arrays and array items for existence with isset before using them.

[up](http://php.net/manual/vote-note.php?id=114671&page=language.variables.basics&vote=up)

[down](http://php.net/manual/vote-note.php?id=114671&page=language.variables.basics&vote=down)

2

[***jeff12 at fastpitchcentral dot com*** ¶](http://php.net/manual/en/language.variables.basics.php#114671)

**3 months ago**

While the recommendation here is to initialize variables, there is a very good reason to definitely initialize variables.  
  
I just had the unpleasant task of making numerous folks happy that the Apache "error\_log" is now much smaller.  We had over 1500 variables that showed up as "error" in the error\_log file.  In many cases just one uninitialized variable might cause 10s or 100s of thousands of records to be entered in the "error\_log".  
  
While it may not be necessary to initialize variables, there can be a significant cost if you do not do so.  
  
And no, that website was not willing to lower the bar and not write to the "error\_log" file if it was simply a case of uninitialized variables.  It was also the case that in a small percentage of cases  the variables newly initialized caused me to scratch my head and conclude that a blank or zero might cause different logic to occur.

[up](http://php.net/manual/vote-note.php?id=107080&page=language.variables.basics&vote=up)

[down](http://php.net/manual/vote-note.php?id=107080&page=language.variables.basics&vote=down)

6

[***megan at voices dot com*** ¶](http://php.net/manual/en/language.variables.basics.php#107080)

**2 years ago**

"Note: $this is a special variable that can't be assigned."  
  
While the PHP runtime generates an error if you directly assign $this in code, it doesn't for $$name when name is 'this'.  
  
<?php  
  
$this = 'text'; // error  
  
$name = 'this';  
$$name = 'text'; // sets $this to 'text'  
  
?>

[up](http://php.net/manual/vote-note.php?id=101978&page=language.variables.basics&vote=up)

[down](http://php.net/manual/vote-note.php?id=101978&page=language.variables.basics&vote=down)

4

[***maurizio dot domba at pu dot t-com dot hr*** ¶](http://php.net/manual/en/language.variables.basics.php#101978)

**3 years ago**

If you need to check user entered value for a proper PHP variable naming convention you need to add ^ to the above regular expression so that the regular expression should be '^[a-zA-Z\_\x7f-\xff][a-zA-Z0-9\_\x7f-\xff]\*'.  
  
Example  
  
<?php  
$name="20011aa";  
if(!preg\_match('/[a-zA-Z\_\x7f-\xff][a-zA-Z0-9\_\x7f-\xff]\*/',$name))  
   echo $name.' is not a valid PHP variable name';  
else  
   echo $name.' is valid PHP variable name';  
?>  
  
Outputs: 2011aa is valid PHP variable name  
  
but  
  
<?php  
$name="20011aa";  
if(!preg\_match('/^[a-zA-Z\_\x7f-\xff][a-zA-Z0-9\_\x7f-\xff]\*/',$name))  
   echo $name.' is not a valid PHP variable name';  
else  
   echo $name.' is valid PHP variable name';  
?>  
  
Outputs: 2011aa is not a valid PHP variable name

[up](http://php.net/manual/vote-note.php?id=96594&page=language.variables.basics&vote=up)

[down](http://php.net/manual/vote-note.php?id=96594&page=language.variables.basics&vote=down)

-3

[***Edoxile*** ¶](http://php.net/manual/en/language.variables.basics.php#96594)

**4 years ago**

When wanting to switch two variables from content, you can use the XOR operator:  
  
<?PHP  
$a=5;  
$b=3;  
  
//Please mind the order of these, as it's important for the outcome.  
  
$a^=$b;  
$b^=$a;  
$a^=$b;  
  
echo $a.PHP\_EOL.$b;  
/\* prints:  
3  
5  
\*/  
?>  
  
This will also work on strings, but it won't work on arrays and objects, so for them you'll have to use the serialize() function before the operation, and the unserialize() function after.

[up](http://php.net/manual/vote-note.php?id=113070&page=language.variables.basics&vote=up)

[down](http://php.net/manual/vote-note.php?id=113070&page=language.variables.basics&vote=down)

-5

[***php at richardneill dot org*** ¶](http://php.net/manual/en/language.variables.basics.php#113070)

**10 months ago**

Note that "$1" is not a variable name. PHP treats it literally, even when it is in double quotes. Eg:  
  
$fruit="apple";  
echo "This $fruit costs $1 ";  
  
This is especially notable when using $1, $2 etc inside parameterised queries in SQL.

[up](http://php.net/manual/vote-note.php?id=113265&page=language.variables.basics&vote=up)

[down](http://php.net/manual/vote-note.php?id=113265&page=language.variables.basics&vote=down)

-8

[***tymac at hotmail dot com*** ¶](http://php.net/manual/en/language.variables.basics.php#113265)

**9 months ago**

Hi,  
  
something like $foo = &myfunc(); seems to work fine.  
  
Regards.

I needed a simple function that would reduce any kind of variable to a string or number while retaining some semblance of the data that was stored in the variable. This is what I came up with:  
  
<?  
function ReduceVar ($Value) {  
    switch (gettype($Value)) {  
        case "boolean":  
        case "integer":  
        case "double":  
        case "string":  
        case "NULL":  
            return $Value;  
        case "resource":  
            return get\_resource\_type($Value);  
        case "object":  
            return ReduceVar(get\_object\_vars($Value));  
        case "array":  
            if (count($Value) <= 0)  
                return NULL;  
            else  
                return ReduceVar(reset($Value));  
        default:  
            return NULL;  
    }  
}  
?>

[up](http://php.net/manual/vote-note.php?id=15605&page=ref.var&vote=up)

[down](http://php.net/manual/vote-note.php?id=15605&page=ref.var&vote=down)

0

[***skelley at diff dot nl*** ¶](http://php.net/manual/en/ref.var.php#15605)

**12 years ago**

Sorry to say Mykolas, but your definition would not be correct.   
  
isempty() evaluates to true for NULL, 0, "", false or 'not set' for any variable, object etc. that can be set to a value.   
  
isset() evaluates to true if the variable, object etc. exists at all, whether it is 'empty' or not.   
  
Example:   
$foo = 0;   
isset($foo); //will evaluate to true.   
!empty($foo); //will evaluate to false.   
  
unset($foo);   
isset($foo); //will evaluate to false.   
!empty($foo); //will evaluate to false.

[up](http://php.net/manual/vote-note.php?id=97138&page=ref.var&vote=up)

[down](http://php.net/manual/vote-note.php?id=97138&page=ref.var&vote=down)

-1

[***coding-kid at hotmail dot com*** ¶](http://php.net/manual/en/ref.var.php#97138)

**4 years ago**

A simple method to check what type input var is:  
  
<code>  
class filter{  
    /\*  
     \* @var: $types (array): holds all the supported datatypes, their number, and the callback that need to be made  
     \*                             to check if it's valid.  
     \*/  
    private $types = array('string'=> array('number'=> '1', 'callback'=> 'is\_string'),  
                            'int'=> array('number' => '2', 'callback' => 'is\_int'),  
                            'array'=> array('number' => '3', 'callback' => 'is\_array'),  
                            'object'=> array('number' => '4', 'callback' => 'is\_object'),  
                            'float'=> array('number' => '5', 'callback' => 'is\_float'),  
                            'bool'=> array('number' => '6', 'callback' => 'is\_bool'),  
                            'resource'=> array('number' => '7', 'callback' => 'is\_resource'),  
                            'null'=> array('number' => '8', 'callback' => 'is\_null'));  
  
    function \_\_construct(){  
          
    }  
    /\*  
     \* @desc: This method checks what type the input variable is  
     \*  
     \* @param: $var (mixed): The var that will be checked  
     \*  
     \* @param: $output\_type (string): How the result will be returned  
     \*  
     \* @returns: string or int  
     \*/  
    function type($var, $output\_type = 'numeric'){  
  
        if(is\_string($var)){  
            $var\_type = array("string", "1");  
        } elseif(is\_int($var)){  
            $var\_type = array("int", "2");  
        } elseif(is\_array($var)){  
            $var\_type = array("array", "3");  
        } elseif(is\_object($var)){  
            $var\_type = array("object", "4");  
        } elseif(is\_float($var)){  
            $var\_type = array("float", "5");  
        } elseif(is\_bool($var)){  
            $var\_type = array("bool", "6");  
        } elseif(is\_resource($var)){  
            $var\_type = array("resource", "7");  
        } elseif(is\_null($var)){  
            $var\_type = array("null", "8");  
        }  
        if($output\_type == 'text'){  
            return $var\_type[0];  
        } else{  
            return $var\_type[1];  
        }  
    }  
}</code>

If you're running PHP as a shell script, and you want to use the argv and argc arrays to get command-line arguments, make sure you have register\_argc\_argv  =  on.  If you're using the 'optimized' php.ini, this defaults to off.

[up](http://php.net/manual/vote-note.php?id=30678&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=30678&page=language.variables.predefined&vote=down)

4

[***LouisGreen at pljg dot freeserve dot co dot uk*** ¶](http://php.net/manual/en/language.variables.predefined.php#30678)

**11 years ago**

It seems that when you wish to export a varible, you can do it as return $varible, return an array(), or globalise it. If you return something, information for that varible can only travel one way when the script is running, and that is out of the function.   
  
function fn() {  
   $varible = "something";  
  
  return $variable;  
}  
  
echo fn();  
OR  
$newvariable = fn();  
  
Although if global was used, it creates a pointer to a varible, whether it existed or not, and makes whatever is created in the function linked to that global pointer. So if the pointer was global $varible, and then you set a value to $varible, it would then be accessible in the global scope. But then what if you later on in the script redefine that global to equal something else. This means that whatever is put into the global array, the information that is set in the pointer, can be set at any point (overiden). Here is an example that might make this a little clearer:  
  
function fn1() {  
  
   global $varible; // Pointer to the global array  
   $varible = "something";  
}  
  
fn1();  
echo $varible; // Prints something  
$varible = "12345";  
echo $varible; // Prints 12345  
  
function fn2() {  
  
   global $varible; // Pointer to the global array  
   echo $varible;  
}  
  
fn2(); // echos $varible which contains "12345"  
  
Basically when accessing the global array, you can set it refer to something already defined or set it to something, (a pointer) such as varible you plan to create in the function, and later possibly over ride the pointer with something else.

[up](http://php.net/manual/vote-note.php?id=64336&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=64336&page=language.variables.predefined&vote=down)

3

[***johnphayes at gmail dot com*** ¶](http://php.net/manual/en/language.variables.predefined.php#64336)

**8 years ago**

I haven't found it anywhere else in the manual, so I'll make a note of it here - PHP will automatically replace any dots ('.') in an incoming variable name with underscores ('\_'). So if you have dots in your incoming variables, e.g.:  
  
example.com/page.php?chuck.norris=nevercries  
  
you can not reference them by the name used in the URI:  
//INCORRECT  
echo $\_GET['chuck.norris'];  
  
instead you must use:  
//CORRECT  
echo $\_GET['chuck\_norris'];

[up](http://php.net/manual/vote-note.php?id=30278&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=30278&page=language.variables.predefined&vote=down)

5

[***LouisGreen at pljg dot freeserve dot co dot uk*** ¶](http://php.net/manual/en/language.variables.predefined.php#30278)

**11 years ago**

If you require access to Predefined Variables in different PHP/ servers versions and don't wish to mess about with how you access them, this little snippet of code might help you:  
  
function fn\_http\_vars\_access() {  
  
   global $GET\_VARS, $POST\_VARS, $COOKIE\_VARS, $SESSION\_VARS, $SERVER\_VARS, $ENV\_VARS;  
  
   $parser\_version = phpversion();  
  
   if ($parser\_version <= "4.1.0") {   
      $GET\_VARS      = $GET\_VARS;  
      $POST\_VARS     = $POST\_VARS;  
      $COOKIE\_VARS   = $COOKIE\_VARS;  
      $SESSION\_VARS  = $HTTP\_SESSION\_VARS;  
      $SERVER\_VARS   = $HTTP\_SERVER\_VARS;  
      $ENV\_VARS      = $HTTP\_ENV\_VARS;  
   }  
   if ($parser\_version >= "4.1.0") {   
      $GET\_VARS      = $\_GET;  
      $POST\_VARS     = $\_POST;  
      $COOKIE\_VARS   = $\_COOKIE;  
      $SESSION\_VARS  = $\_SESSION;  
      $SERVER\_VARS   = $\_SERVER;  
      $ENV\_VARS      = $\_ENV;  
   }  
}  
  
fn\_http\_vars\_access();

[up](http://php.net/manual/vote-note.php?id=80794&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=80794&page=language.variables.predefined&vote=down)

2

[***root at mantoru dot de*** ¶](http://php.net/manual/en/language.variables.predefined.php#80794)

**6 years ago**

To tokie at hanmail dot net: You took that out of context -- it is merely a recommendation.  
  
If your variables\_order setting does not contain "E", $\_ENV is still useful. Every call to getenv will be "cached" in $\_ENV, so you can do this:  
  
<?php  
// variables\_order = GPCS  
var\_dump(isset($\_ENV['PATH'])); // bool(false)  
getenv('PATH');  
var\_dump(isset($\_ENV['PATH'])); // bool(true)  
?>  
  
For some reason, it does not work with with own environment variables. The above example with PHP\_TEST instead of PATH would fail (if it is set via putenv).

[up](http://php.net/manual/vote-note.php?id=63708&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=63708&page=language.variables.predefined&vote=down)

1

[***jk at ricochetsolutions dot com*** ¶](http://php.net/manual/en/language.variables.predefined.php#63708)

**8 years ago**

here is a one line snippet to do the same as DD32's func  
  
@preg\_replace(  
   "/(?i)([a-z0-9\_]+)\/([a-z0-9\_]+)\/?/e",   
   '$\_GET[\'$1\'] = "$2";',   
   ((isset($\_SERVER['PATH\_INFO'])) ? $\_SERVER['PATH\_INFO'] : '')  
);  
  
may be faster, it may not ;o

[up](http://php.net/manual/vote-note.php?id=63312&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=63312&page=language.variables.predefined&vote=down)

1

[***DD32=theonly\_DD32[&amp;]yahoo.com.au*** ¶](http://php.net/manual/en/language.variables.predefined.php#63312)

**8 years ago**

I have this function in my main files, it allows for easier SEO for some pages without having to rely on .htaccess and mod\_rewrite for some things.  
<?php  
    function long\_to\_GET(){  
        /\*\*  
        \* This function converts info.php/a/1/b/2/c?d=4 TO  
        \* Array ( [d] => 4 [a] => 1 [b] => 2 [c] => )   
        \*\*/  
        if(isset($\_SERVER['PATH\_INFO']) && $\_SERVER['PATH\_INFO'] != ''){  
            //Split it out.  
            $tmp = explode('/',$\_SERVER['PATH\_INFO']);  
            //Remove first empty item  
            unset($tmp[0]);  
            //Loop through and apend it into the $\_GET superglobal.  
            for($i=1;$i<=count($tmp);$i+=2){ $\_GET[$tmp[$i]] = $tmp[$i+1];}  
        }  
    }  
?>  
  
Its probably not the most efficient, but it does the job rather nicely.  
  
DD32

[up](http://php.net/manual/vote-note.php?id=31772&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=31772&page=language.variables.predefined&vote=down)

1

[***alexsp at olywa dot net*** ¶](http://php.net/manual/en/language.variables.predefined.php#31772)

**11 years ago**

For those of us who don't have the luxery of upgrading to the latest version of PHP on all of the servers we use but want to use the same variable names that are used in the latest version for super global arrays here's a snippet of code that will help:  
    // Makes available those super global arrays that are made available  
    // in versions of PHP after v4.1.0.  
    if (isset ($HTTP\_SERVER\_VARS))  
    {  
        $\_SERVER = &$HTTP\_SERVER\_VARS;  
    }  
      
    if (isset ($HTTP\_GET\_VARS))  
    {  
        $\_GET = &$HTTP\_GET\_VARS;  
    }  
      
    if (isset ($HTTP\_POST\_VARS))  
    {  
        $\_POST = &$HTTP\_POST\_VARS;  
    }  
      
    if (isset ($HTTP\_COOKIE\_VARS))  
    {  
        $\_COOKIE = &$HTTP\_COOKIE\_VARS;  
    }  
      
    if (isset ($HTTP\_POST\_FILES))  
    {  
        $\_FILES = &$HTTP\_POST\_FILES;  
    }  
      
    if (isset ($HTTP\_ENV\_VARS))  
    {  
        $\_ENV = &$HTTP\_ENV\_VARS;  
    }  
      
    if (isset ($HTTP\_SESSION\_VARS))  
    {  
        $\_SESSION = &$HTTP\_SESSION\_VARS;  
    }  
The only downfall to this is that there's no way to make them super global. Chances are, though, if you're using a lot of global arrays in your code you should consider a code redesign!  :)  Hope this helps.

[up](http://php.net/manual/vote-note.php?id=52304&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=52304&page=language.variables.predefined&vote=down)

2

[***myfirstname dot barros at gmail dot com*** ¶](http://php.net/manual/en/language.variables.predefined.php#52304)

**9 years ago**

vars in $\_REQUEST are \*not\* a reference to the respective $\_POST and $\_GET and $\_COOKIE ones.  
  
Consider:  
<http://site.com/index.php?avar=abc>  
  
index.php:  
<?php  
$\_GET['avar'] = 'b';  
print\_r($\_GET); print('<br>');  
print\_r($\_REQUEST);  
?>  
  
output:  
Array ( [avar] => 'b' )  
Array ( [avar] => 'abc' )

[up](http://php.net/manual/vote-note.php?id=52857&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=52857&page=language.variables.predefined&vote=down)

2

[***dompody [at] gmail [dot] com*** ¶](http://php.net/manual/en/language.variables.predefined.php#52857)

**9 years ago**

To urbanheroes:  
  
version\_compare() is only in PHP version 4.1.0 and up. This completely negates your function, since if the version is less than 4.1.0 it will generate an error anyway. The better solution is to do what is stated in the post above yours:  
  
<?php  
if (!isset($\_SERVER))  
{  
   $\_GET    = &$HTTP\_GET\_VARS;  
   $\_POST    = &$HTTP\_POST\_VARS;  
   $\_ENV    = &$HTTP\_ENV\_VARS;  
   $\_SERVER  = &$HTTP\_SERVER\_VARS;  
   $\_COOKIE  = &$HTTP\_COOKIE\_VARS;  
   $\_REQUEST = array\_merge($\_GET, $\_POST, $\_COOKIE);  
}  
?>  
  
Include that before everything else in your script and it will fix the flaw.

[up](http://php.net/manual/vote-note.php?id=72829&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=72829&page=language.variables.predefined&vote=down)

1

[***holger at doessing dot net*** ¶](http://php.net/manual/en/language.variables.predefined.php#72829)

**7 years ago**

On the subject of permalinks and queries:  
Say, you use an inexpensive subdomain of (e.g.) www.nice.net, thus www.very.nice.net, and that the domain owner has simply placed a frame at this particular location, linking to the actual address (ugly and subject-to-change) of your site.  
Consequently, the actual site URI and various associated hashes and query strings are not immediately visible to the user. Sometimes this is useful, but it also makes bookmarking/permalinking impossible (the browser will only bookmark the static address in the top frame).  
However, as far as the query strings go, there is workaround. Instead of providing users with permalinks to the actual URI (e.g. prtcl://weird.and.ugly/~very/ugly.php?stuff=here; may even be subject to change), I provide them with this: prtcl://www.very.nice.net?stuff=here.  
  
In brief, I then use the following code to re-populate the $\_GET array:  
  
if (isset($\_SERVER['HTTP\_REFERER'])) { // If set, this page is running in a frame  
    $uri = parse\_url($\_SERVER['HTTP\_REFERER']); // grab URI of parent frame  
    $querystring = ($uri['query']) ? $uri['query'] : false; // grab the querystring  
    if ($querystring) {  
        $vars = explode('&', $querystring); // cut into individual statements  
        foreach ($vars as $varstring) { // populate $\_GET  
            $var = explode('=', $varstring);  
            if (count($var) == 2) $\_GET[$var[0]] = $var[1];  
        }  
    } // no, nothing to report from the parent frame  
} // no, not using a parent frame today...  
  
If the actual host address is ever changed, users entering the frame (with the nicer address) will be using the new (and ugly) URI, but this way the old query strings will be available to the new address also. The users will never again be bothered by you moving to another neighborhood.

[up](http://php.net/manual/vote-note.php?id=74620&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=74620&page=language.variables.predefined&vote=down)

1

[***pinkgothic at gmail dot com*** ¶](http://php.net/manual/en/language.variables.predefined.php#74620)

**7 years ago**

Dealing with "superglobals" and functions is not as straightforward as it may seem when you're doing plenty manipulations.  
  
For example:  
  
<?php  
  function some\_other\_method() {  
    echo $\_REQUEST['id'];  
  }  
  function some\_method() {  
    $\_REQUEST['id'] = 440;  
    some\_other\_method();  
  }  
?>  
  
Calling some\_method() will cause a warning-level error by PHP informing you that "id" is not set in some\_other\_method(). However, if you instead use:  
  
<?php  
  $\_REQUEST['id'] = 0;  
  function some\_other\_method() {  
    echo $\_REQUEST['id'];  
  }  
  function some\_method() {  
    $\_REQUEST['id'] = 440;  
    some\_other\_method();  
  }  
?>  
  
Then the script will echo 440.  
  
In consequence, if you manually attempt to add keys to the superglobals, those keys \*aren't\* automatically superglobal. The above example isn't very sensible, of course, but this can be a huge gotcha if you're juggling user data between functions and you're unwittingly being forced to work inside a function (e.g. via PHP include in TYPO3).  
  
Unfortunately, global $\_REQUEST['id'] won't save you, either - it causes a parse error - nor will a global $\_REQUEST change anything after you've set the keys... consequently making it hard to conviniently 'hack' outdated scripts by making them believe they're still running in a different environment.  
  
The only "solution" to this issue is to use parameters.

[up](http://php.net/manual/vote-note.php?id=68765&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=68765&page=language.variables.predefined&vote=down)

0

[***yarco dot w at gmail dot com*** ¶](http://php.net/manual/en/language.variables.predefined.php#68765)

**7 years ago**

And you should know  
  
$\_POST is not a reference of $HTTP\_POST\_VARS  
  
So, if you change $\_POST, there are no change to $HTTP\_POST\_VARS.

[up](http://php.net/manual/vote-note.php?id=65659&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=65659&page=language.variables.predefined&vote=down)

0

[***Iñigo Medina*** ¶](http://php.net/manual/en/language.variables.predefined.php#65659)

**8 years ago**

It is true. I usually write variables in this way: $chuckNorrisFilms. So one almost never finds problems.

[up](http://php.net/manual/vote-note.php?id=62735&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=62735&page=language.variables.predefined&vote=down)

0

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.predefined.php#62735)

**8 years ago**

there is a difference to the scope of eg. java: variables that are defined inside a block are also defined outside of  the brackets.  
  
eg. this works:  
  
if {true}  
{  
  $a = 'it works';  
}  
  
echo $a;

[up](http://php.net/manual/vote-note.php?id=56717&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=56717&page=language.variables.predefined&vote=down)

0

[***Graeme Jefferis*** ¶](http://php.net/manual/en/language.variables.predefined.php#56717)

**8 years ago**

I find this sort of thing consistently useful for dealing with superglobals in safety and comfort.  
<?php  
foreach ($\_POST as $key => $value)  
{  
        switch ($key)  
        {  
                case "submitted\_var\_1":  
                case "submitted\_var\_2":  
                case "submitted\_var\_3":  
                        $$key = $value; break;  
  
                case "dangerous\_var":  
                        $value = do\_something\_special\_with($value);  
                        $$key = $value;  
                        break;  
        }  
}  
?>

[up](http://php.net/manual/vote-note.php?id=56689&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=56689&page=language.variables.predefined&vote=down)

0

[***Nicole King*** ¶](http://php.net/manual/en/language.variables.predefined.php#56689)

**8 years ago**

There seems to a maximum size of key that you can use for the $\_SESSION array on php5. If you exceed this length, which seems to be around 72 characters, the value is stored in the array, but is not serialised and restored later in the session (ie. when a subsquent page is processed). The same restriction \*might\* apply to other system-defined arrays.

[up](http://php.net/manual/vote-note.php?id=56289&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=56289&page=language.variables.predefined&vote=down)

0

[***webdesign at benking dot com*** ¶](http://php.net/manual/en/language.variables.predefined.php#56289)

**8 years ago**

# this is a follow-up to kasey at cornerspeed's 14-Jun-2004 08:33 post and debabratak at softhome's 14-Mar-2003 12:59 post, minus sessions but including a safety mechanism to block unwanted variables...  
  
# if you are like me and do not want to have to type $\_POST[some\_var] to get to all your passed variable data, you can safely convert all the data to the variable names (so it is like old style php) by using a pre-defined allowed arg names list like this;  
  
$allowed\_args = ',f\_name,l\_name,subject,msg,';  
  
foreach(array\_keys($\_POST) as $k) {  
    $temp = ",$k,";  
    if(strpos($allowed\_args,$temp) !== false) { $$k = $\_POST[$k]; }  
}  
  
# then you can use the programmer friendly (less typing) vars like so;  
echo "Hello $f\_name";  
  
# make sure you have commas in front of and after each var name in the $allowed\_args list, so strpos will never surprise you by mistakingly finding an unwanted var name within another var name

[up](http://php.net/manual/vote-note.php?id=52283&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=52283&page=language.variables.predefined&vote=down)

0

[***sendoshin[at]noodleroni[dot]com*** ¶](http://php.net/manual/en/language.variables.predefined.php#52283)

**9 years ago**

There is one way to safely execute PHP code files without running the risk of compromising your own code.  A prior note pointed out that the code being evaluated would still have access to globals using the global keyword.  While this is a valid point, there's one other approach to be looked at - one which actually gives you much more ability than just unsetting some variable references.  It's known as code parsing.  
  
The specifics would be different and much more complex in a deployed site, but here's an extremely strip-down example of how to restrict access to global variables:  
  
<?php  
    while ($x = stristr ($code\_to\_eval, "global")) {  
        $temp = substr ($code\_to\_eval, 1, $x-1);  
        $temp .= substr ($code\_to\_eval, stristr ($code\_to\_eval, ";", $x) + 1);  
        $code\_to\_eval = $temp;  
    }  
    $ret\_val = eval ($code\_to\_eval);  
?>  
  
Of course, that's just a rudimentary example, and a deployment version would have much more checking involved, but parsing the file before you eval it lets you remove any code you don't want to let run, thus making it as safe as your parsing rules.

[up](http://php.net/manual/vote-note.php?id=51757&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=51757&page=language.variables.predefined&vote=down)

0

[***lanny at freemail dot hu*** ¶](http://php.net/manual/en/language.variables.predefined.php#51757)

**9 years ago**

From PHP 5.0.3 long predefined arrays such HTTP\_GET\_VARS got disabled by default. For backward compatibility you can enable them in php.ini:  
  
register\_long\_arrays = On  
  
I sugget a big WARNING up there like that one with the resister\_globals.   
  
Anyway.. I cannot understand why they do such tings all the time.

[up](http://php.net/manual/vote-note.php?id=50044&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=50044&page=language.variables.predefined&vote=down)

0

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.predefined.php#50044)

**9 years ago**

php.net uses this  
  
// Backward compatible array creation. After this point, the  
// PHP 4.1.0+ arrays can be used to access variables coming  
// from outside PHP. But it should be noted that these variables  
// are not necessarily superglobals, so they need to be global-ed!  
if (!isset($\_SERVER))  
{  
    $\_GET     = &$HTTP\_GET\_VARS;  
    $\_POST    = &$HTTP\_POST\_VARS;  
    $\_ENV     = &$HTTP\_ENV\_VARS;  
    $\_SERVER  = &$HTTP\_SERVER\_VARS;  
    $\_COOKIE  = &$HTTP\_COOKIE\_VARS;  
    $\_REQUEST = array\_merge($\_GET, $\_POST, $\_COOKIE);  
}  
  
$PHP\_SELF = $\_SERVER['PHP\_SELF'];

[up](http://php.net/manual/vote-note.php?id=43228&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=43228&page=language.variables.predefined&vote=down)

0

[***kasey at cornerspeed dowt com*** ¶](http://php.net/manual/en/language.variables.predefined.php#43228)

**10 years ago**

I have a few points to note to (debabratak at softhome dot net).  Firstly, extracting all your variables from the global variable arrays is rather cumbersome and possibly unsafe.  This causes longer run times, and wastes more memory.  Then, your script is starting the session before it parses the superglobals.  Bad things can happen because of this:  
  
<?php  
  
// user sent a GET header with key = secret\_access, val = true, so  
  
echo $\_GET["secret\_access"]; // output: true  
echo $secret\_access; // output:  
  
session\_start();  
  
// in previous logic, you set session variable $secret\_access = false  
  
echo $\_SESSION["secret\_access"]; // output: false  
echo $secret\_access; // output: false  
  
extract\_globals();  // Globals put into "normal" variables  
  
echo $\_GET["secret\_access"]; // output: true  
echo $\_SESSION["secret\_access"]; // output: false  
echo $secret\_access; // output: true  
  
// VARIABLES ARE COMPROMISED!  
// DO NOT USE $secret\_access !  
// USE $\_SESSION["secret\_access"] instead !!!  
  
?>  
  
Secondly, I would like to point out the fact that all $\_POST, $\_GET, and $\_COOKIE variables are intrinsically unsafe anyway.  Users can create their own scripts in the language of their choosing (PHP, ASP, JSP, etc.) that generate those headers to send to your PHP program via socket connections.  PHP cannot determine that these headers are any less valid than the ones sent by a web browser, so it parses them and places them in the $\_POST, $\_GET, or $\_COOKIE variables.  
  
The best practice is to use $\_SESSION variables to validate the user before making any decisions based on form data.  e.g.:  
  
<?php  
session\_start();  
if (isset($\_SESSION["valid"]))  
{  
    // all your program decisions and database interactions can go here  
    if (isset($\_POST["button\_name"]))  
    {  
        ...  
    }  
    ...  
}  
elseif (isset($\_POST["submit\_login"]))  
{  
    if (($\_POST["username"] == "foo") AND ($\_POST["password"] == "bar"))  
    {  
        $\_SESSION["valid"] = true;  
        ...  
    }  
    else  
    {  
        session\_unset();  
        session\_destroy();  
        $error\_msg = "Invalid username or password";  
        $result\_page = "login.php";  
    }  
}  
elseif (isset($logoff))  
{  
    session\_unset();  
    session\_destroy();  
    $success\_msg = "You have logged off successfully";  
    $result\_page = "login.php";  
}  
else  
{  
    session\_unset();  
    session\_destroy();  
    $result\_page = "login.php";  
}  
require ($result\_page);  
?>  
  
Session variables are orders of magnitude harder to compromise than POST, GET, and COOKIE data, since the server keeps track of session id's, and the session id is unique to each client and somewhat randomly generated.  If security is an ultimate concern, then you need to use SSL in case your traffic can be sniffed (since the session cookie is passed plain text to the client).  
  
In summary, extracting out all the superglobals to normal variable names is not a good idea for reasons of security and ambiguity, not to mention wasted CPU cycles.  For private applications (ones that you don't want just anyone to be able to access), the only ways you can prevent malicious access is to 1) use sessions to ensure that the user is valid (for that page), and 2) use SSL-encryption to prevent session-hijacking.  
  
Kasey  
  
in reply to:  
--------------------------------------------------------------  
debabratak at softhome dot net  
14-Mar-2003 12:59  
After having register\_globals = off, I am using the following piece of code to get all the variables created for me. I have put this code in a separate file and just make it require\_once() on top of every page.  
  
session\_start();  
$ArrayList = array("\_GET", "\_POST", "\_SESSION", "\_COOKIE", "\_SERVER");  
foreach($ArrayList as $gblArray)  
{  
   $keys = array\_keys($$gblArray);  
   foreach($keys as $key)  
   {  
       $$key = trim(${$gblArray}[$key]);  
   }  
}  
  
This pulls out all the possible variables for me, including the predefined variables, so I can keep coding the old style. Note that, this code does not handle the $\_FILE.  
  
Hope this helps someone.

[up](http://php.net/manual/vote-note.php?id=42270&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=42270&page=language.variables.predefined&vote=down)

0

[***bryan at nolifeline dot com*** ¶](http://php.net/manual/en/language.variables.predefined.php#42270)

**10 years ago**

to marcbender at mail dot com  
  
unset the globals  
  
use a preg\_replace ( pattern: |\;[^\;]\*$i[^\;]\*\;|Uis, replacement: ";", where $i is the name of any function/variable you wish to prevent access to.) on the code-to-be-evaled.  ideas are "global", "fopen", "mysql\_connect", etc.  You know, anything that you wouldn't want to give a hyperactive 13 year old access to.  
  
execute the code.

[up](http://php.net/manual/vote-note.php?id=41656&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=41656&page=language.variables.predefined&vote=down)

0

[***marcbender\_AT\_mail\_DOT\_com*** ¶](http://php.net/manual/en/language.variables.predefined.php#41656)

**10 years ago**

-Security issue-  
  
In response to lopez at yellowspace,  
  
You provided a method for executing potentially unsafe code:  
  
> function safeEval($evalcode) {  
>    unset($GLOBALS);  
>    unset($\_ENV);  
>    // unset any other superglobal...  
>    return eval($evalcode);  
> }  
  
Your method, though clever, won't work.  The problem is the way that PHP handles function scope.  If $evalcode contains a function declaration, and runs that function, the "unset"s will be effectively useless inside the body of that function.  
  
Try running the above code with $evalcode set as follows:  
  
<?php  
$evalcode='f();  
function f() {  
   $GLOBALS["\_SERVER"] = "compromised";  
}';  
?>  
  
Then print $\_SERVER and see what you get.  
  
Another problem is that the "global" directive will always grant access to global variables.  Try this one:  
  
<?php  
$evalcode='global $a;  
$a = "compromised";';  
?>  
  
$a will of course be changed at the global level.  I don't know if it's supposed to work this way, but on my system (PHP 4.3.4) you can do the same to any superglobal by importing it using "global".  
  
As far as I can tell, there is NO way to execute potentially unsafe code without a lot of risk.  With the sloppy way that PHP deals with function scope etc., there isn't much hope that it ever will be.  What we'd need is (at least):  
  - a way to disable the "global" directive (restrictive eval).  
  - a way to shut off any write-access to superglobals within untrusted functions.  
  
The first wouldn't be too hard to implement.  The second, on the other hand, is practically impossible IMHO.

[up](http://php.net/manual/vote-note.php?id=36878&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=36878&page=language.variables.predefined&vote=down)

0

[***mark at pitchpipe dot org*** ¶](http://php.net/manual/en/language.variables.predefined.php#36878)

**10 years ago**

I had always mistakenly assumed that superglobal $\_COOKIE (while preferred) was identical to the outdated $HTTP\_COOKIE\_VARS.  However, if you assign:  
  
$\_COOKIE['destroyWorld'] = "true";  
if (isset($HTTP\_COOKIE\_VARS['destroyWorld'])) {  
   $temp =& new Armeggedon();  
   $temp->pushRedButton();  
}  
  
then the world will be safe forever.  Might throw off a newbie, or someone like me who was updating really old code bit-by-bit.

[up](http://php.net/manual/vote-note.php?id=36167&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=36167&page=language.variables.predefined&vote=down)

0

[***wagner at cesnet dot cz*** ¶](http://php.net/manual/en/language.variables.predefined.php#36167)

**10 years ago**

The redirected pages by response codes 301, 302, 303 change the request method always to GET, that's why $HTTP\_POST\_VARS are lost. It is described in Apache documentation.

[up](http://php.net/manual/vote-note.php?id=35627&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=35627&page=language.variables.predefined&vote=down)

0

[***joker at vip dot hr*** ¶](http://php.net/manual/en/language.variables.predefined.php#35627)

**10 years ago**

If anyone of you have a problem with uploading files with globals off here is the solution... just add this to the top of the code:  
  
reset ($\_FILES);  
while (list ($key, $val) = each ($\_FILES)) {  
    ${$key}=$\_FILES[$key]['tmp\_name'];  
    while (list ($key1, $val1) = each ($val)) {  
        ${$key."\_".$key1}=$\_FILES[$key][$key1];  
    }  
}  
  
   Daniel

[up](http://php.net/manual/vote-note.php?id=31724&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=31724&page=language.variables.predefined&vote=down)

0

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.predefined.php#31724)

**11 years ago**

In reply to destes at ix dot netcom dot com dot nospam:  
  
It's possible for a HTTP client to spoof HTTP\_X\_FORWARDED\_FOR, and set it to a fake IP number.  It's more secure to use this code and log BOTH the ip and the proxy ip.  
  
if ($\_SERVER["HTTP\_X\_FORWARDED\_FOR"]) {  
   if ($\_SERVER["HTTP\_CLIENT\_IP"]) {  
    $proxy = $\_SERVER["HTTP\_CLIENT\_IP"];  
  } else {  
    $proxy = $\_SERVER["REMOTE\_ADDR"];  
  }  
  $ip = $\_SERVER["HTTP\_X\_FORWARDED\_FOR"];  
} else {  
  if ($\_SERVER["HTTP\_CLIENT\_IP"]) {  
    $ip = $\_SERVER["HTTP\_CLIENT\_IP"];  
  } else {  
    $ip = $\_SERVER["REMOTE\_ADDR"];  
  }  
}  
  
echo "Your IP $ip<BR>\n";  
if (isset($proxy)) {  
  echo "Your proxy IP is $proxy<BR>\n";  
}

[up](http://php.net/manual/vote-note.php?id=30484&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=30484&page=language.variables.predefined&vote=down)

0

[***Good Liam*** ¶](http://php.net/manual/en/language.variables.predefined.php#30484)

**11 years ago**

Warning:  
If you use dynamic variables in a local scope, the variable doesn't "know" when it should be a superglobal.  An example will help elucidate this:  
  
function Example($Variable\_Name='\_POST') {  
    print\_r($$Variable\_Name);  
} // End Example  
  
This would print out  
  
NULL  
  
To use a dynamic variable to reference a superglobal, you have to declare the value (not the name) as a global:  
  
function WorkingExample($Variable\_Name='\_POST') {  
    global $$Variable\_Name;  
    print\_r($$Variable\_Name);  
} // End WorkingExample()  
  
This would print out the contents of your $\_POST variable.  
  
This threw me when I first tried it, but it makes sense, in a way.

[up](http://php.net/manual/vote-note.php?id=29386&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=29386&page=language.variables.predefined&vote=down)

0

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.predefined.php#29386)

**11 years ago**

i just noticed that the free web server i'm running my scripts on still only knows the deprecated variable names (i.e. it uses $HTTP\_POST\_VARS instead of $\_POST). to make scripts work both on updated servers and servers that are a bit out of date, i now use:  
  
$variablename=(isset($\_POST["variablename"])) ? $\_POST["variablename"] : $HTTP\_POST\_VARS["variablename"];

[up](http://php.net/manual/vote-note.php?id=28614&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=28614&page=language.variables.predefined&vote=down)

0

[***lopez dot on dot the dot lists at yellowspace dot net*** ¶](http://php.net/manual/en/language.variables.predefined.php#28614)

**11 years ago**

- Security Issue and workaround -   
If You use "eval()" to execute code stored in a database or elsewhere, you might find this tip useful.  
  
Issue:  
By default, all superglobals are known in every function.   
Thus, if you eval database- or dynamically generated code (let's call it "potentially unsafe code"), it can use \_all\_ the values stored in \_any\_ superglobal.   
  
Workaround:  
Whenever you want to hide superglobals from use in evaluated code, wrap that eval() in an own function within which you unset() all the superglobals. The superglobals are not deleted by php in all scopes - just within that function. eg:  
  
function safeEval($evalcode) {  
    unset($GLOBALS);  
    unset($\_ENV);  
    // unset any other superglobal...  
    return eval($evalcode);  
}  
  
(This example assumes that the eval returns something with 'return')  
  
In addition, by defining such a function outside classes, in the global scope, you'll make sure as well that the evaluated ('unsafe') code doesn't have access to the object variables ($this-> ...).

[up](http://php.net/manual/vote-note.php?id=23063&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=23063&page=language.variables.predefined&vote=down)

0

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.predefined.php#23063)

**12 years ago**

Wouldn't it be great if there was a variable called $\_SERVER["PATH\_USERHOME"]. Here is how to set it yourself:  
  
$path\_fs = split ("/", ltrim ($\_SERVER["PATH\_TRANSLATED"], "/"));  
$path\_fs\_rev = array\_reverse ($path\_fs);  
  
$path\_http = split ("/", ltrim ($\_SERVER["PHP\_SELF"], "/"));  
$path\_http\_rev = array\_reverse ($path\_http);  
  
$num\_same = 0;  
while ($path\_fs\_rev[$num\_same] == $path\_http\_rev[$num\_same]) {  
    $num\_same++;  
}  
  
$path\_userhome = array ();  
$numdirs\_userhome = sizeof ($path\_http) - $num\_same;  
echo $numdirs\_userhome;  
  
for ($i = 0; $i < $numdirs\_userhome; $i++) {  
    array\_push ($path\_userhome, $path\_http[$i]);  
}  
  
$\_SERVER["PATH\_USERHOME"] = "/" . implode ("/", $path\_userhome) . "/";  
  
print\_r ($\_SERVER["PATH\_USERHOME"]);  
  
;) Happy programming,  
  
Peder

[up](http://php.net/manual/vote-note.php?id=21952&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=21952&page=language.variables.predefined&vote=down)

0

[***juancri at TAGnet dot org*** ¶](http://php.net/manual/en/language.variables.predefined.php#21952)

**12 years ago**

If you try this:  
  
<FORM action="hola">  
  ....  
</FORM>  
  
and hola is a directory, you have to write the final slash (/) because the page is redirected from hola to hola/ and you'll lost the POST variables.

[up](http://php.net/manual/vote-note.php?id=14246&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=14246&page=language.variables.predefined&vote=down)

0

[***rick@independence,netI*** ¶](http://php.net/manual/en/language.variables.predefined.php#14246)

**12 years ago**

It should be noted that $HTTP\_RAW\_POST\_DATA only exists if the encoding type of the data is -not- the default of application/x-www.form-urlencoded, and so, to accessing raw post data from an HTTP form requires setting enctype= in your HTML.

[up](http://php.net/manual/vote-note.php?id=76062&page=language.variables.predefined&vote=up)

[down](http://php.net/manual/vote-note.php?id=76062&page=language.variables.predefined&vote=down)

-1

[***fabrizio at bibivu dot com*** ¶](http://php.net/manual/en/language.variables.predefined.php#76062)

**7 years ago**

theonly\_DD32, I refined your function a little bit  
  
<?php  
    function long\_to\_GET($PATH\_INFO=''){  
        /\*\*  
        \* This function converts info.php/a/1/b/2/c?d=4 TO  
        \* array ( [d] => 4 [a] => 1 [b] => 2 [c] => array ( [d] => 4 ) )  
        \* got this function from <http://php.net/GLOBALS>  
        \*\*/  
        if($PATH\_INFO=='' && isset($\_SERVER['PATH\_INFO']) && $\_SERVER['PATH\_INFO'] != ''){  
            $PATH\_INFO = $\_SERVER['PATH\_INFO'];  
        }  
        if($PATH\_INFO != ''){  
            //Split it out.  
            $tmp = explode('/',$PATH\_INFO);  
            //Remove first empty item  
            unset($tmp[0]);  
            //Loop through and apend it into the $\_GET superglobal.  
            for($i=1;$i<=count($tmp);$i+=2){  
                if(strpos($tmp[$i],'?')!==false){  
                    $tmp1 = explode('?',$tmp[$i]);  
                    parse\_str(isset($tmp1[1])?$tmp1[1]:'',$\_GET[$tmp1[0]]);  
                    $i--;  
                } else {  
                    $\_GET[$tmp[$i]] = isset($tmp[$i+1])?$tmp[$i+1]:'';  
                }  
            }  
        }  
    }  
  
?>

Some interesting behavior (tested with PHP5), using the static-scope-keyword inside of class-methods.  
  
<?php  
  
class sample\_class  
{  
  public function func\_having\_static\_var($x = NULL)  
  {  
    static $var = 0;  
    if ($x === NULL)  
    { return $var; }  
    $var = $x;  
  }  
}  
  
$a = new sample\_class();  
$b = new sample\_class();  
  
echo $a->func\_having\_static\_var()."\n";  
echo $b->func\_having\_static\_var()."\n";  
// this will output (as expected):  
//  0  
//  0  
  
$a->func\_having\_static\_var(3);  
  
echo $a->func\_having\_static\_var()."\n";  
echo $b->func\_having\_static\_var()."\n";  
// this will output:  
//  3  
//  3  
// maybe you expected:  
//  3  
//  0  
  
?>  
  
One could expect "3 0" to be outputted, as you might think that $a->func\_having\_static\_var(3); only alters the value of the static $var of the function "in" $a - but as the name says, these are class-methods. Having an object is just a collection of properties, the functions remain at the class. So if you declare a variable as static inside a function, it's static for the whole class and all of its instances, not for each object.  
  
Maybe it's senseless to post that.. cause if you want to have the behaviour that I expected, you can simply use a variable of the object itself:  
  
<?php  
class sample\_class  
{ protected $var = 0;   
  function func($x = NULL)  
  { $this->var = $x; }  
} ?>  
  
I believe that all normal-thinking people would never even try to make this work with the static-keyword, for those who try (like me), this note maybe helpfull.

[up](http://php.net/manual/vote-note.php?id=89872&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=89872&page=language.variables.scope&vote=down)

4

[***Leigh Harrison*** ¶](http://php.net/manual/en/language.variables.scope.php#89872)

**5 years ago**

External variables in a function   
  
I needed to access dynamically-created variables from an included file within a helper function. Because the list of $path\_\* variables I needed to access from the other file is itself dynamic, I didn't want to have to declare all possible variables within the function, and I was concerned at the overhead of declaring =all= members of $GLOBALS[] as global. However the following code worked for me:   
  
<?php   
  function makePath($root, $atom) {   
    $pos = strrpos($atom, '/');   
    if ($pos === false) {   
      global ${'path\_'.$atom};    
      $path = ${'path\_'.$atom};   
    }   
    else {   
      global ${'path\_'.substr($atom, 0, $pos)};   
      $path = ${'path\_'.substr($atom, 0, $pos)};   
    }   
    if ($path)   
      return ($pos === false)   
        ? $root.$path   
        : $root.$path.substr($atom, $pos + 1);   
    else   
      return NULL;   
  }   
?>   
  
Regards,   
  
::Leigh

[up](http://php.net/manual/vote-note.php?id=100884&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=100884&page=language.variables.scope&vote=down)

7

[***php at keith tyler dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#100884)

**3 years ago**

Sometimes a variable available in global scope is not accessible via the 'global' keyword or the $GLOBALS superglobal array. I have not been able to replicate it in original code, but it occurs when a script is run under PHPUnit.  
  
PHPUnit provides a variable "$filename" that reflects the name of the file loaded on its command line. This is available in global scope, but not in object scope. For example, the following phpUnit script (call it GlobalScope.php):  
  
<?php  
print "Global scope FILENAME [$filename]\n";  
class MyTestClass extends PHPUnit\_Framework\_TestCase {  
  function testMyTest() {  
    global $filename;  
    print "Method scope global FILENAME [$filename]\n";  
    print "Method scope GLOBALS[FILENAME] [".$GLOBALS["filename"]."]\n";  
  }  
}  
?>  
  
If you run this script via "phpunit GlobalScope.php", you will get:  
  
Global scope FILENAME [/home/ktyler/GlobalScope.php]  
PHPUnit 3.4.5 by Sebastian Bergmann.  
  
Method scope global FILENAME []  
Method scope GLOBALS[FILENAME] []  
.  
  
You have to -- strange as it seems -- do the following:  
  
<?php  
$GLOBALS["filename"]=$filename;  
print "Global scope FILENAME [$filename]\n";  
class MyTestClass extends PHPUnit\_Framework\_TestCase {  
  function testMyTest() {  
    global $filename;  
    print "Method scope global FILENAME [$filename]\n";  
    print "Method scope GLOBALS[FILENAME] [".$GLOBALS["filename"]."]\n";  
  }  
}  
?>  
  
By doing this, both "global" and $GLOBALS work!  
  
I don't know what it is that PHPUnit does (I know it uses Reflection) that causes a globally available variable to be implicitly unavailable via "global" or $GLOBALS. But there it is.

[up](http://php.net/manual/vote-note.php?id=98577&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=98577&page=language.variables.scope&vote=down)

5

[***jakub dot lopuszanski at nasza-klasa dot pl*** ¶](http://php.net/manual/en/language.variables.scope.php#98577)

**4 years ago**

If you use \_\_autoload function to load classes' definitons, beware that "static local variables are resolved at compile time" (whatever it really means) and the order in which autoloads occur may impact the semantic.  
  
For example if you have:  
<?php  
class Singleton{  
  static public function get\_instance(){  
     static $instance = null;  
     if($instance === null){  
        $instance = new static();  
     }  
     return $instance;  
  }  
}  
?>  
  
and two separate files A.php and B.php:  
class A extends Singleton{}  
class B extends A{}  
  
then depending on the order in which you access those two classes, and consequently, the order in which \_\_autoload includes them, you can get strange results of calling B::get\_instance() and A::get\_instance().  
  
It seems that static local variables are alocated in as many copies as there are classes that inherit a method at the time of inclusion of parsing Singleton.

[up](http://php.net/manual/vote-note.php?id=88689&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=88689&page=language.variables.scope&vote=down)

5

[***andrew at planetubh dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#88689)

**5 years ago**

Took me longer than I expected to figure this out, and thought others might find it useful.  
  
I created a function (safeinclude), which I use to include files; it does processing before the file is actually included (determine full path, check it exists, etc).  
  
Problem: Because the include was occurring inside the function, all of the variables inside the included file were inheriting the variable scope of the function; since the included files may or may not require global variables that are declared else where, it creates a problem.  
  
Most places (including here) seem to address this issue by something such as:  
<?php  
//declare this before include  
global $myVar;  
//or declare this inside the include file  
$nowglobal = $GLOBALS['myVar'];  
?>  
  
But, to make this work in this situation (where a standard PHP file is included within a function, being called from another PHP script; where it is important to have access to whatever global variables there may be)... it is not practical to employ the above method for EVERY variable in every PHP file being included by 'safeinclude', nor is it practical to staticly name every possible variable in the "global $this" approach. (namely because the code is modulized, and 'safeinclude' is meant to be generic)  
  
My solution: Thus, to make all my global variables available to the files included with my safeinclude function, I had to add the following code to my safeinclude function (before variables are used or file is included)  
  
<?php  
foreach ($GLOBALS as $key => $val) { global $$key; }  
?>  
  
Thus, complete code looks something like the following (very basic model):  
  
<?php  
function safeinclude($filename)  
{  
    //This line takes all the global variables, and sets their scope within the function:  
    foreach ($GLOBALS as $key => $val) { global $$key; }  
    /\* Pre-Processing here: validate filename input, determine full path  
        of file, check that file exists, etc. This is obviously not  
        necessary, but steps I found useful. \*/  
    if ($exists==true) { include("$file"); }  
    return $exists;  
}  
?>  
  
In the above, 'exists' & 'file' are determined in the pre-processing. File is the full server path to the file, and exists is set to true if the file exists. This basic model can be expanded of course.  In my own, I added additional optional parameters so that I can call safeinclude to see if a file exists without actually including it (to take advantage of my path/etc preprocessing, verses just calling the file exists function).  
  
Pretty simple approach that I could not find anywhere online; only other approach I could find was using PHP's eval().

[up](http://php.net/manual/vote-note.php?id=98811&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=98811&page=language.variables.scope&vote=down)

10

[***HOSSEIN doesn&#39;t want spam at TAKI.IR*** ¶](http://php.net/manual/en/language.variables.scope.php#98811)

**4 years ago**

Please note for using global variable in child functions:   
  
This won't work correctly...   
  
<?php   
function foo(){   
    $f\_a = 'a';   
      
    function bar(){   
        global $f\_a;   
        echo '"f\_a" in BAR is: ' . $f\_a . '<br />';  // doesn't work, var is empty!   
    }   
      
    bar();   
    echo '"f\_a" in FOO is: ' . $f\_a . '<br />';   
}   
?>   
  
This will...   
  
<?php   
function foo(){   
    global $f\_a;   // <- Notice to this   
    $f\_a = 'a';   
      
    function bar(){   
        global $f\_a;   
        echo '"f\_a" in BAR is: ' . $f\_a . '<br />';  // work!, var is 'a'   
    }   
      
    bar();   
    echo '"f\_a" in FOO is: ' . $f\_a . '<br />';   
}   
?>

[up](http://php.net/manual/vote-note.php?id=92868&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=92868&page=language.variables.scope&vote=down)

6

[***Stephen Dewey*** ¶](http://php.net/manual/en/language.variables.scope.php#92868)

**4 years ago**

For nested functions:  
  
This is probably obvious to most people, but global always refers to the variable in the global (top level) variable of that name, not just a variable in a higher-level scope. So this will not work:  
<?php  
  
// $var1 is not declared in the global scope  
  
function a($var1){  
  
    function b(){  
        global $var1;  
        echo $var1; // there is no var1 in the global scope so nothing to echo  
      
    }  
  
    b();  
}  
  
a('hello');  
  
?>

[up](http://php.net/manual/vote-note.php?id=81506&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=81506&page=language.variables.scope&vote=down)

7

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.scope.php#81506)

**6 years ago**

I was pondering a little something regarding caching classes within a function in order to prevent the need to initiate them multiple times and not clutter the caching function's class properties with more values.  
  
I came here because I remembered something about references being lost. So I made a test to see if I could pull what I wanted to off anyway. Here's and example of how to get around the references lost issue. I hope it is helpful to someone else!  
  
<?php  
class test1{}  
class test2{}  
class test3{}  
  
function cache( $class )  
{  
    static $loaders = array();  
      
    $loaders[ $class ] = new $class();  
  
    var\_dump( $loaders );  
}  
print '<pre>';  
cache( 'test1' );  
cache( 'test2' );  
cache( 'test3' );  
  
?>

[up](http://php.net/manual/vote-note.php?id=85203&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=85203&page=language.variables.scope&vote=down)

3

[***ddarjany at yahoo dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#85203)

**5 years ago**

Note that if you declare a variable in a function, then set it as global in that function, its value will not be retained outside of that function.  This was tripping me up for a while so I thought it would be worth noting.  
  
<?PHP  
  
foo();  
echo $a; // echoes nothing  
  
bar();  
echo $b; //echoes "b";  
  
function foo() {  
  $a = "a";   
  global $a;  
}  
  
function bar() {  
  global $b;  
  $b = "b";  
}  
  
?>

[up](http://php.net/manual/vote-note.php?id=113913&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=113913&page=language.variables.scope&vote=down)

1

[***Jonathan Kenigson*** ¶](http://php.net/manual/en/language.variables.scope.php#113913)

**6 months ago**

Just a note about static properties declared at class level:  
  
class Test\_Class {  
  static $a = 0;  
  public function ReturnVar(){  
    return $this->a;  
  }  
  }  
  $b = new Test\_Class();  
  echo $b->ReturnVar();  
  
Will not output "0"  because $a is declared static. Changing "static" to "public" or "private" will produce the output "0".

[up](http://php.net/manual/vote-note.php?id=18748&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=18748&page=language.variables.scope&vote=down)

1

[***admin at essentialhost dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#18748)

**12 years ago**

Quick tip for beginners just to speed things up:   
If you have a bunch of global variables to import into a function, it's best to put them into a named array like $variables[stuff].   
When it's time to import them you just so the following;   
  
<?php   
function here() {   
  $vars = $GLOBALS['variables'];   
  print $vars[stuff];   
  
}   
?>   
  
This really helps with big ugly form submissions.

[up](http://php.net/manual/vote-note.php?id=109310&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=109310&page=language.variables.scope&vote=down)

2

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.scope.php#109310)

**2 years ago**

It will be obvious for most of you: changing value of a static in one instance changes value in all instances.  
  
<?php  
  
    class example {  
        public static $s = 'unchanged';  
          
        public function set() {  
            $this::$s = 'changed';  
        }  
    }  
  
    $o = new example;  
    $p = new example;  
  
    $o->set();  
  
    print "$o static: {$o::$i}\n$p static: {$p::$i}";  
  
?>  
  
Output will be:  
  
$o static: changed  
$p static: changed

[up](http://php.net/manual/vote-note.php?id=14251&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=14251&page=language.variables.scope&vote=down)

1

[***danno at wpi dot edu*** ¶](http://php.net/manual/en/language.variables.scope.php#14251)

**12 years ago**

WARNING!  If you create a local variable in a function and then within that function assign it to a global variable by reference the object will be destroyed when the function exits and the global var will contain NOTHING!  This main sound obvious but it can be quite tricky you have a large script (like a phpgtk-based gui app ;-) ).   
  
example:   
  
<?php   
function foo ()   
{   
   global $testvar;   
  
   $localvar = new Object ();   
   $testvar = &$localvar;   
}   
  
foo ();   
print\_r ($testvar);   // produces NOTHING!!!!   
?>   
  
hope this helps someone before they lose all their hair

[up](http://php.net/manual/vote-note.php?id=56749&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=56749&page=language.variables.scope&vote=down)

1

[***tc underline at gmx TLD ch*** ¶](http://php.net/manual/en/language.variables.scope.php#56749)

**8 years ago**

Pay attention while unsetting variables inside functions:  
  
<?php  
$a = "1234";  
echo "<pre>";  
echo "outer: $a\n";  
function testa()  
{  
    global $a;  
    echo "   inner testa: $a\n";  
    unset ($a);  
    echo "   inner testa: $a\n";  
}  
function testb()  
{  
    global $a;  
    echo "   inner testb: $a\n";  
    $a = null;  
    echo "   inner testb: $a\n";  
}  
testa();  
echo "outer: $a\n";  
testb();  
echo "outer: $a\n";  
echo "</pre>";  
?>  
  
/\*\*\*\*\* Result:  
outer: 1234  
   inner testa: 1234  
   inner testa:   
outer: 1234  
   inner testb: 1234  
   inner testb:   
outer:   
\*\*\*\*\*\*/  
  
Took me 1 hour to find out why my variable was still there after unsetting it ...  
  
Thomas Candrian

[up](http://php.net/manual/vote-note.php?id=105925&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=105925&page=language.variables.scope&vote=down)

1

[***dodothedreamer at gmail dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#105925)

**2 years ago**

Note that unlike Java and C++, variables declared inside blocks such as loops or if's, will also be recognized and accessible outside of the block, so:  
<?php  
for($j=0; $j<3; $j++)  
{  
     if($j == 1)  
        $a = 4;  
}  
echo $a;  
?>  
  
Would print 4.

[up](http://php.net/manual/vote-note.php?id=27546&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=27546&page=language.variables.scope&vote=down)

1

[***wjs@sympaticoDOTca*** ¶](http://php.net/manual/en/language.variables.scope.php#27546)

**11 years ago**

Becareful where you define your global variables:   
  
This will work:  
<?php   
  $MyArray = array("Dog");  
  
  function SeeArray(){  
    global $MyArray;  
    if (in\_array("Dog",$MyArray)){  
      foreach ($MyArray as $Element){  
        echo "$Element <hr/>";  
      }  
    }  
  }  
  
  SeeArray();  
?>  
  
while this will not:  
<?php   
  SeeArray();  
  $MyArray = array("Dog");  
  
  function SeeArray(){  
    global $MyArray;  
    if (in\_array("Dog",$MyArray)){ // an error will generate here  
      foreach ($MyArray as $Element){  
        echo "$Element <hr/>";  
      }  
    }  
  }  
  
?>

[up](http://php.net/manual/vote-note.php?id=109479&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=109479&page=language.variables.scope&vote=down)

0

[***pangxiezhou at gmail dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#109479)

**1 year ago**

assign after static variable declare：  
  
<?php   
        function test() {  
                static $local\_var=0;  
                $local\_var=9;  
                echo $local\_var++;  
            }  
        test();  
        test();  
          
?>  
  
output:  
99  
  
<?php   
        function test() {  
                static $local\_var=0;  
                $local\_var+=3;  
                echo $local\_var++;  
            }  
        test();  
        test();  
          
?>  
  
output:  
37

[up](http://php.net/manual/vote-note.php?id=100312&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=100312&page=language.variables.scope&vote=down)

0

[***eduardo dot ferron at zeion dot net*** ¶](http://php.net/manual/en/language.variables.scope.php#100312)

**3 years ago**

There're times when global variables comes in handy, like universal read only resources you just need to create once in your application and share to the rest of your scripts. But it may become quite hard to track with "variables".

[up](http://php.net/manual/vote-note.php?id=99074&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=99074&page=language.variables.scope&vote=down)

0

[***pedro at worcel dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#99074)

**3 years ago**

Another way of working with a large ammount of global variables could be the following.  
  
<?php  
  
$var = "3";  
$smarty = new Smarty();  
  
function headers\_set\_404() {  
extract($globals);  
  
echo $var . "<br />";  
print\_r($smarty);  
  
return;  
  
}  
  
?>  
  
Regards,  
Droope

[up](http://php.net/manual/vote-note.php?id=94259&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=94259&page=language.variables.scope&vote=down)

0

[***moraesdno at gmail dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#94259)

**4 years ago**

Use the superglobal array $GLOBALS is faster than the global keyword. See:   
  
<?php   
//Using the keyword global   
$a=1;   
$b=2;   
function sum() {   
    global $a, $b;   
    $a += $b;   
}   
  
$t = microtime(true);   
for($i=0; $i<1000; $i++) {   
     sum();   
}   
echo microtime(true)-$t;   
echo " -- ".$a."<br>";   
  
//Using the superglobal array   
$a=1;   
$b=2;   
function sum2() {   
    $GLOBALS['a'] += $GLOBALS['b'];   
}   
  
  $t = microtime(true);   
for($i=0; $i<1000; $i++) {   
     sum2();   
}   
echo microtime(true)-$t;   
echo " -- ".$a."<br>";   
?>

[up](http://php.net/manual/vote-note.php?id=92166&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=92166&page=language.variables.scope&vote=down)

0

[***akam at akameng dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#92166)

**4 years ago**

Many Times Globality of variables will be the small issue, after long time I decided to use super globals.  
  
Super globals exists any where:  
$\_SERVER, $\_GET, $\_POST .....  
  
Now for example:  
  
<?php  
$foo[] = range(0, 3);  
$\_POST['foo'] = $foo;  
a(); //no parameters needed.  
b();  
$foo = $\_POST['foo'];  
  
Print\_r($foo);  
/\* out  
  
Array  
(  
    [0] => Array  
        (  
            [0] => 0  
            [1] => 1  
            [2] => 2  
            [3] => 3  
        )  
  
    [1] => Array  
        (  
            [0] => 4  
            [1] => 5  
            [2] => 6  
            [3] => 7  
        )  
  
    [2] => Array  
        (  
            [0] => 8  
            [1] => 9  
            [2] => 10  
        )  
  
)  
  
\*/  
function a(){  
    $\_POST['foo'][] = range(4, 7);  
}  
  
function b(){  
$\_POST['foo'][] = range(8, 10);  
}  
?>  
Note: the key must not be passed by the page via \_POST method by the form, else the value will be over written

[up](http://php.net/manual/vote-note.php?id=88570&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=88570&page=language.variables.scope&vote=down)

0

[***nullhility at gmail dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#88570)

**5 years ago**

Like functions, if you declare a variable in a class, then set it as global in that class, its value will not be retained outside of that class either.   
  
<?php   
class global\_reference   
{   
    public $val;   
      
    public function \_\_construct () {   
        global $var;   
        $this->val = $var;   
    }   
      
    public function dump\_it ()   
    {   
        debug\_zval\_dump($this->val);   
    }   
      
    public function type\_cast ()   
    {   
        $this->val = (int) $this->val;   
    }   
}   
$var = "x";   
$obj = new global\_reference();   
$obj->dump\_it();   
$obj->type\_cast();   
echo "after change ";   
$obj->dump\_it();   
echo "original $var\n";   
?>   
  
The work-around is of course changing the assignment in the constructor to a reference assignment as such:   
  
<?php   
    //....   
        $this->val = &var;   
    //....   
?>   
  
If the global you're setting is an object then no reference is necessary because of the way PHP deals with objects. If you don't want to reference to the same object however you can use the clone keyword.   
  
<?php   
//...   
    global $Obj;   
    $this->obj\_copy = clone $Obj;   
//...   
?>   
  
[EDIT BY danbrown AT php DOT net:  Merged all thoughts and notes by this author into a single note.]

[up](http://php.net/manual/vote-note.php?id=83490&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=83490&page=language.variables.scope&vote=down)

0

[***lgrk*** ¶](http://php.net/manual/en/language.variables.scope.php#83490)

**6 years ago**

Useful function:  
<?php  
function cycle($a, $b, $i=0) {  
    static $switches = array();  
    if (isset($switches[$i])) $switches[$i] = !$switches[$i]; else !$switches[$i] = true;  
    return ($switches[$i])?$a:$b;  
}  
?>  
  
Exeample  
  
<?php  
for ($i = 1; $i<3; $i++) {  
    echo $i.cycle('a', 'b').PHP\_EOL;  
    for ($j = 1; $j<5; $j++) {  
        echo ' '.$j.cycle('a', 'b', 1).PHP\_EOL;  
        for ($k = 1; $k<3; $k++) {  
            echo '  '.$k.cycle('c', 'd', 2).PHP\_EOL;  
        }  
    }  
}  
/\*\*  
Output:  
1a  
1a  
  1c  
  2d  
2b  
  1c  
  2d  
3a  
  1c  
  2d  
4b  
  1c  
  2d  
2b  
1a  
  1c  
  2d  
2b  
  1c  
  2d  
3a  
  1c  
  2d  
4b  
  1c  
  2d  
\*/  
  
?>

[up](http://php.net/manual/vote-note.php?id=113384&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=113384&page=language.variables.scope&vote=down)

-1

[***Tomas Binek*** ¶](http://php.net/manual/en/language.variables.scope.php#113384)

**9 months ago**

Be careful when using autoloaders:  
Any variables that are declared in autoloaded files in global context are in fact local variables to the autoload function.  
Suggestion:  
Use explicit global keyword even in global context of a file.

[up](http://php.net/manual/vote-note.php?id=69611&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=69611&page=language.variables.scope&vote=down)

0

[***alan*** ¶](http://php.net/manual/en/language.variables.scope.php#69611)

**7 years ago**

Using the global keyword inside a function to define a variable is essentially the same as passing the variable by reference as a parameter:   
  
<?php   
somefunction(){   
   global $var;   
}   
?>   
  
is the same as:   
  
<?php   
somefunction(& $a) {   
  
}   
?>   
  
The advantage to using the keyword is if you have a long list of variables  needed by the function - you dont have to pass them every time you call the function.

[up](http://php.net/manual/vote-note.php?id=113115&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=113115&page=language.variables.scope&vote=down)

-1

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.scope.php#113115)

**10 months ago**

<?php  
  
// if you really want to create a variable within its own scope   
// that does not have access to variables outside its scope create a function  
  
$var = "hello";  
  
$func = function(){  
  
      // declare variables here that will only last throughout this scope  
  
     if( !isset($var) ) // var will not be set in this scope  
     {  
        $var = "i was out of scope";  
     }  
        
      echo $var;  
  
};  
  
echo "$var<br />";  
  
$func(); // invoke the function  
  
echo "<br />".'$var'." never changed from $var";  
  
?>  
  
outputs :  
  
hello  
i was out of scope  
$var never changed from hello

[up](http://php.net/manual/vote-note.php?id=63500&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=63500&page=language.variables.scope&vote=down)

0

[***larax at o2 dot pl*** ¶](http://php.net/manual/en/language.variables.scope.php#63500)

**8 years ago**

About more complex situation using global variables..  
  
Let's say we have two files:  
a.php  
<?php   
    function a() {   
        include("b.php");   
    }  
    a();  
?>  
  
b.php  
<?php  
    $b = "something";  
    function b() {  
        global $b;  
        $b = "something new";  
    }  
    b();  
    echo $b;  
?>  
  
You could expect that this script will return "something new" but no, it will return "something". To make it working properly, you must add global keyword in $b definition, in above example it will be:  
  
global $b;  
$b = "something";

[up](http://php.net/manual/vote-note.php?id=60231&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=60231&page=language.variables.scope&vote=down)

0

[***marcin*** ¶](http://php.net/manual/en/language.variables.scope.php#60231)

**8 years ago**

Sometimes in PHP 4 you need static variabiles in class. You can do it by referencing static variable in constructor to the class variable:  
  
<?php  
class test  {  
  
   var $var;  
   var $static\_var;  
    function test()  
    {  
        static $s;  
        $this->static\_var =& $s;  
    }  
    
}  
  
$a=new test();  
  
$a->static\_var=4;  
$a->var=4;  
  
$b=new test();  
  
echo $b->static\_var; //this will output 4  
echo $b->var; //this will output nul  
?>

[up](http://php.net/manual/vote-note.php?id=55595&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=55595&page=language.variables.scope&vote=down)

0

[***thomas at pixtur dot de*** ¶](http://php.net/manual/en/language.variables.scope.php#55595)

**8 years ago**

Be careful with "require", "require\_once" and "include" inside functions. Even if the included file seems to define global variables, they might not be defined as such.   
  
consider those two files:   
  
---index.php------------------------------   
<?php   
function foo() {   
require\_once("class\_person.inc");   
  
$person= new Person();   
echo $person->my\_flag; // should be true, but is undefined   
}   
  
foo();   
?>   
  
---class\_person.inc----------------------------   
<?php   
$seems\_global=true;   
  
class Person {   
  public $my\_flag;   
  
public function  \_\_construct() {   
   global $seems\_global;   
   $my\_flag= $seems\_global   
}   
}   
?>   
  
---------------------------------   
  
The reason for this behavior is quiet obvious, once you figured it out. Sadly this might not be always as easy as in this example. A solution  would be to add the line...   
  
<?php global $seems\_global; ?>   
  
at the beginning of "class\_person.inc". That makes sure you set the global-var.   
  
   best regards   
    tom   
  
ps: bug search time approx. 1 hour.

[up](http://php.net/manual/vote-note.php?id=53915&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=53915&page=language.variables.scope&vote=down)

0

[***jameslee at cs dot nmt dot edu*** ¶](http://php.net/manual/en/language.variables.scope.php#53915)

**9 years ago**

It should be noted that a static variable inside a method is static across all instances of that class, i.e., all objects of that class share the same static variable.  For example the code:  
  
<?php  
class test {  
    function z() {  
        static $n = 0;  
        $n++;  
        return $n;  
    }  
}  
  
$a =& new test();  
$b =& new test();  
print $a->z();  // prints 1, as it should  
print $b->z();  // prints 2 because $a and $b have the same $n  
?>  
  
somewhat unexpectedly prints:  
1  
2

[up](http://php.net/manual/vote-note.php?id=52343&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=52343&page=language.variables.scope&vote=down)

0

[***kouber at php dot net*** ¶](http://php.net/manual/en/language.variables.scope.php#52343)

**9 years ago**

If you need all your global variables available in a function, you can use this:   
  
<?php   
function foo() {   
  extract($GLOBALS);   
  // here you have all global variables   
  
}   
?>

[up](http://php.net/manual/vote-note.php?id=52294&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=52294&page=language.variables.scope&vote=down)

0

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.scope.php#52294)

**9 years ago**

Be careful if your static variable is an array and you return  
one of it's elements: Other than a scalar variable, elements  
of an array are returned as reference (regardless if you   
didn't define them to be returned by reference).  
  
<?php  
function incr(&$int) {  
  return $int++;  
}  
  
function return\_copyof\_scalar() {  
  static $v;  
  if (!$v)     
    $v = 1;  
  return($v);   
}   
  
function return\_copyof\_arrayelement() {  
  static $v;  
  if (!$v) {  
    $v = array();  
    $v[0] = 1;  
  }  
  return($v[0]);   
}   
  
echo "scalar: ".  
     incr(return\_copyof\_scalar()).  
     incr(return\_copyof\_scalar()).  
     "\n";   
echo "arrayelement: ".  
     incr(return\_copyof\_arrayelement()).  
     incr(return\_copyof\_arrayelement()).  
     "\n";   
?>  
  
Should print  
  
scalar: 11  
arrayelement: 11  
  
but it prints:  
  
scalar: 11  
arrayelement: 12  
  
as in the second case the arrays element was returned by  
reference. According to a guy from the bug reports the   
explanation for this behaviour should be somewhere here in   
the documentation (in 'the part with title: "References with   
global and static variables"'). Unfortunately I can't find   
anything about that here. As the guys from the bug reports   
are surely right in every case, maybe there is something   
missing in the documentation. Sadly I don't have a good   
explanation why this happens, so I decided to document at   
least the behaviour.

[up](http://php.net/manual/vote-note.php?id=45282&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=45282&page=language.variables.scope&vote=down)

0

[***info AT SyPlex DOT net*** ¶](http://php.net/manual/en/language.variables.scope.php#45282)

**9 years ago**

Some times you need to access the same static in more than one function. There is an easy way to solve this problem:  
  
<?php  
  // We need a way to get a reference of our static  
  function &getStatic() {  
    static $staticVar;  
    return $staticVar;  
  }  
  
  // Now we can access the static in any method by using it's reference  
  function fooCount() {  
    $ref2static = & getStatic();  
    echo $ref2static++;  
  }  
  
  fooCount(); // 0  
  fooCount(); // 1  
  fooCount(); // 2  
?>

[up](http://php.net/manual/vote-note.php?id=42955&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=42955&page=language.variables.scope&vote=down)

0

[***Michael Bailey (jinxidoru at byu dot net)*** ¶](http://php.net/manual/en/language.variables.scope.php#42955)

**10 years ago**

Static variables do not hold through inheritance.  Let class A have a function Z with a static variable.  Let class B extend class A in which function Z is not overwritten.  Two static variables will be created, one for class A and one for class B.  
  
Look at this example:  
  
<?php  
class A {  
    function Z() {  
        static $count = 0;          
        printf("%s: %d\n", get\_class($this), ++$count);  
    }  
}  
  
class B extends A {}  
  
$a = new A();  
$b = new B();  
$a->Z();  
$a->Z();  
$b->Z();  
$a->Z();  
?>  
  
This code returns:  
  
A: 1  
A: 2  
B: 1  
A: 3  
  
As you can see, class A and B are using different static variables even though the same function was being used.

[up](http://php.net/manual/vote-note.php?id=41213&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=41213&page=language.variables.scope&vote=down)

0

[***Randolpho*** ¶](http://php.net/manual/en/language.variables.scope.php#41213)

**10 years ago**

More on static variables:   
  
A static variable does not retain it's value after the script's execution. Don't count on it being available from one page request to the next; you'll have to use a database for that.   
  
Second, here's a good pattern to use for declaring a static variable based on some complex logic:   
  
<?php   
  function buildStaticVariable()   
  {   
      $foo = null;   
      // some complex expression or set of   
      // expressions/statements to build   
      // the return variable.   
      return $foo;   
  }   
  
  function functionWhichUsesStaticVar()   
  {   
      static $foo = null;   
      if($foo === null) $foo = buildStaticVariable();   
      // the rest of your code goes here.   
  }   
?>   
  
Using such a pattern allows you to separate the code that creates your default static variable value from the function that uses it. Easier to maintain code is good. :)

[up](http://php.net/manual/vote-note.php?id=39091&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=39091&page=language.variables.scope&vote=down)

0

[***jmarbas at hotmail dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#39091)

**10 years ago**

Whats good for the goose is not always good for the iterative gander. If you declare and initialize the static variable more than once inside a function ie.   
  
<?php   
function Test(){   
   static $count = 0;   
   static $count = 1;   
   static $count = 2;   
   echo $count;   
}   
?>   
  
the variable will take the value of the last declaration. In this case $count=2.   
  
But! however when you make that function recursive ie.   
  
<?php   
  function Test(){   
   static $count = 0;   
   static $count = 1;   
   static $count = 2;   
  
   $count++;   
   echo $count;   
   if ($count<10){   
     Test();   
   }   
  }   
?>   
  
Every call to the function Test() is a differenct SCOPE and therefore the static declarations and initializations are NOT executed again. So what Im trying to say is that its OK to declare and initialize a static variable multiple times if you are in one function... but its NOT OK to declare and initialize a static variable multiple times if you call that same function multiple times. In other words the static variable is set once you LEAVE a function (even if you go back into that very same function).

[up](http://php.net/manual/vote-note.php?id=37427&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=37427&page=language.variables.scope&vote=down)

0

[***Jack at soinsincere dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#37427)

**10 years ago**

Alright, so you can't set a static variable with a reference.  
However, you can set a static variable to an array with an element that is a reference:  
<?php  
  
class myReference {  
    function getOrSet($array = null) {  
        static $myValue;  
        if (!$array) {  
            return $myValue[0];     //Return reference in array  
        }  
        $myValue = $array;          //Set static variable with array  
        static $myValue;  
    }  
}  
  
$static = "Dummy";  
  
$dummy = new myReference;  
$dummy->getOrSet(array(&$static));  
  
$static = "Test";  
print $dummy->getOrSet();  
  
?>

[up](http://php.net/manual/vote-note.php?id=37183&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=37183&page=language.variables.scope&vote=down)

0

[***flobee at gmx dot net*** ¶](http://php.net/manual/en/language.variables.scope.php#37183)

**10 years ago**

i found out that on any (still not found) reason the <?php static $val =NULL; ?> is not working when trying to extract the data form the $var with a while statment  
e.g.:  
<?php  
funktion get\_data() {  
static $myarray = null;  
   if($myarray == NULL) {  
     //get some info in an array();  
     $myarray = array('one','two');  
   }  
   while(list($key,$val) = each( $myarray ) ) {  
   // do something  
   echo "x: $key , y: $val";  
   }  
}  
?>  
when using foreach($myarray AS $key => $val) { .... instad of while then i see the result!

[up](http://php.net/manual/vote-note.php?id=33832&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=33832&page=language.variables.scope&vote=down)

0

[***ppo at beeznest dot net*** ¶](http://php.net/manual/en/language.variables.scope.php#33832)

**11 years ago**

Even if an included file return a value using return(), it's still sharing the same scope as the caller script!  
  
<?php  
$foo = 'aaa';  
$bar = include('include.php');  
echo($foo.' / '.$bar);  
?>  
  
where include.php is  
<?php  
$foo = 'bbb';  
return $foo;  
?>  
  
The output is: bbb / bbb  
Not: aaa / bbb

[up](http://php.net/manual/vote-note.php?id=12787&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=12787&page=language.variables.scope&vote=down)

0

[***carpathia\_uk at mail dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#12787)

**13 years ago**

On confusing aspect about global scope...   
  
If you want to access a variable such as a cookie inside a function, but theres a chance it may not even be defined, you need to access it using he GLOBALS array, not by defining it as global.   
  
This wont work correctly....   
  
<?php   
function isLoggedin()   
{   
global $cookie\_username;   
if (isset($cookie\_username)   
echo "blah..";   
}   
?>   
  
This will..   
  
<?php   
function isLoggedin()   
{   
if (isset($GLOBALS["cookie\_username"]))   
echo "blah..";   
}   
?>

[up](http://php.net/manual/vote-note.php?id=115085&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=115085&page=language.variables.scope&vote=down)

-2

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.scope.php#115085)

**1 month ago**

Exemplo #6 don't work on portuguese version.  
  
You need to call Teste() function, not Test()

[up](http://php.net/manual/vote-note.php?id=3596&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=3596&page=language.variables.scope&vote=down)

0

[***shevek at anarres dot org*** ¶](http://php.net/manual/en/language.variables.scope.php#3596)

**14 years ago**

If you include a file from within a function using include(), the included file inherits the function scope as its own global scope, it will not be able to see top level globals unless they are explicit in the function.   
  
<?php   
$foo = "bar";   
function baz() {   
    global $foo; # NOTE THIS   
    include("qux");   
}   
?>

[up](http://php.net/manual/vote-note.php?id=29273&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=29273&page=language.variables.scope&vote=down)

-1

[***jg at nerd-boy dot net*** ¶](http://php.net/manual/en/language.variables.scope.php#29273)

**11 years ago**

It's possible to use a variable variable when specifying a variable as global in a function. That way your function can decide what global variable to access in run-time.   
  
<?php   
function func($varname)   
{   
   global $$varname;   
  
   echo $$varname;   
}   
  
$hello = "hello world!";   
func("hello");   
?>   
  
This will print "hello world!", and is roughly the same as passing by reference, in the case when the variable you want to pass is global. The advantage over references is that they can't have default parameters. With the method above, you can do the following.   
  
<?php   
function func($varname = FALSE)   
{   
   if ($varname === FALSE)   
     echo "No variable.";   
   else   
   {   
     global $$varname;   
  
     echo $$varname;   
   }   
}   
  
$hello = "hello world!";   
func("hello");                   // prints "hello world!"   
func();                          // prints "No variable."   
?>

[up](http://php.net/manual/vote-note.php?id=26015&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=26015&page=language.variables.scope&vote=down)

-1

[***heatwave at fw dot hu*** ¶](http://php.net/manual/en/language.variables.scope.php#26015)

**11 years ago**

Some people (including me) had a problem with defining a long GLOBAL variable list in functions (very error prone). Here is a possible solution. My program parses php file for functions, and compiles GLOBAL variable lists. Then you can just remove from the list those variables which need not be global.  
  
<?php  
    //parser for GLOBAL variable list  
    $pfile=file("myfile.php4");  
      
    for($i=0;$i<sizeof($pfile);$i++) {  
     if(eregi("function",$pfile[$i])) {  
      list($part1,$part2)=sscanf($pfile[$i],"%s %s");  
      echo "\n\n $part1 $part2:\nGLOBAL ";  
        
      $varlist=array();  
      $level=0; $end=$i;  
      do {  
       $lpar=explode("{",$pfile[$end]);  
       $level+=sizeof($lpar)-1;  
       $lpar=explode("}",$pfile[$end]);  
       $level-=sizeof($lpar)-1;  
       $end++;  
      } while(($end<sizeof($pfile))&&($level>0));  
      $pstr="";  
      for($j=$i;$j<=$end;$j++) $pstr.=$pfile[$j];  
      $lpar=explode("$",$pstr);  
      for($j=1;$j<sizeof($lpar);$j++) {  
          eregi('[a-zA-Z\_][a-zA-Z0-9\_]\*',$lpar[$j],$cvar);  
        $varlist[$cvar[0]]=1;  
      }  
      array\_walk($varlist,'var\_print');  
     }  
    }  
function var\_print ($item, $key) {  
     echo "$key,";  
}  
?>

[up](http://php.net/manual/vote-note.php?id=21140&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=21140&page=language.variables.scope&vote=down)

-1

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.scope.php#21140)

**12 years ago**

Seems as though when a cookie is saved and referenced as a variable of the same name as the cookie, that variable is NOT global.  If you make a function ro read the value of the cookie, the cooke variable name must be declared as a global.   
  
example:   
  
<?php   
function ReturnCookie()   
{   
        $cookieName = "Test\_Cookie";   
        global $$cookieName;   
        if (isset($$cookieName))   
        {   
                echo ("$cookieName is set");   
                $returnvalue = $$cookieName;   
        }   
        else   
        {   
                $newCookieValue = "Test Value";   
                setcookie("$cookieName","$newCookieValue", (time() + 3153600));   
                echo ("made a cookie:" . $newCookieValue ."<BR>");   
                $returnvalue = $newCookieValue;   
        }   
        echo ("the cookie that was set is now $returnvalue <BR>");   
        return $returnvalue;   
}   
?>

[up](http://php.net/manual/vote-note.php?id=18905&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=18905&page=language.variables.scope&vote=down)

-1

[***steph\_rondinaud at club-internet dot fr*** ¶](http://php.net/manual/en/language.variables.scope.php#18905)

**12 years ago**

I'm using PHP 4.1.1  
  
While designing a database access class, I needed a static variable that will be incremented for all instances of the class each time the class connected to the database. The obvious solution was to declare a "connection" class variable with static scope. Unfortunatly, php doesn't allow such a declaration.  
So I went back to defining a static variable in the connect method of my class. But it seems that the static scope is not inherited: if class "a" inherit the "db access" class, then the "connection" variable is shared among "a" instances, not among both "a" AND "db access" instances.   
Solution is to declare the static variable out of the db access class, and declare "global" said variable in the connect method.

[up](http://php.net/manual/vote-note.php?id=17466&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=17466&page=language.variables.scope&vote=down)

-1

[***tomek at pluton dot pl*** ¶](http://php.net/manual/en/language.variables.scope.php#17466)

**12 years ago**

When defining static variables you may use such declarations:   
  
<?php   
static $var = 1; //numbers   
static $var = 'strings';   
static $var = array(1,'a',3); //array construct   
?>   
  
but these ones would produce errors:   
  
<?php   
static $var = some\_function('arg');   
static $var = (some\_function('arg'));   
static $var = 2+3; //any expression   
static $var = new object;   
?>

[up](http://php.net/manual/vote-note.php?id=45487&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=45487&page=language.variables.scope&vote=down)

-2

[***pulstar at ig dot com dot br*** ¶](http://php.net/manual/en/language.variables.scope.php#45487)

**9 years ago**

If you need all your global variables available in a function, you can use this:  
  
<?php  
  
function foo(parameters) {  
  if(version\_compare(phpversion(),"4.3.0")>=0) {  
    foreach($GLOBALS as $arraykey=>$arrayvalue) {  
      global $$arraykey;  
    }  
  }  
  // now all global variables are locally available...  
}  
  
?>

[up](http://php.net/manual/vote-note.php?id=101699&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=101699&page=language.variables.scope&vote=down)

-1

[***travesty3 at gmail dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#101699)

**3 years ago**

I was struggling forever to figure this out and finally tried the $GLOBALS["filename"] = $filename approach, and it worked for me.   
  
This happens on one of my machines, on which I installed Zend AMF a few months before my most recent machine. The globals worked fine when calling the PHP script directly from a browser, but when I called the function from my Flash project, which uses Zend AMF to interface with the PHP script, I was seeing this problem, but it was fixed using this approach.

[up](http://php.net/manual/vote-note.php?id=61756&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=61756&page=language.variables.scope&vote=down)

-1

[***franp at free dot fr*** ¶](http://php.net/manual/en/language.variables.scope.php#61756)

**8 years ago**

If you want to access a table row using $GLOBALS, you must do it outside string delimiters or using curl braces :   
  
<?php   
$siteParams["siteName"] = "myweb";   
  
function foo() {   
$table = $GLOBALS["siteParams"]["siteName"]."articles";  // OK   
echo $table; // output  "mywebarticles"   
$table = "{$GLOBALS["siteParams"]["siteName"]}articles"; // OK   
echo $table; // output  "mywebarticles"   
$table = "$GLOBALS[siteParams][siteName]articles";       // Not OK   
echo $table; // output  "Array[siteName]article"   
  
$result = mysql\_query("UPDATE $table ...");   
}   
?>   
  
Or use global :   
  
<?php   
function foo() {   
global $siteParams;   
$table = "$siteParams[siteName]articles";         // OK   
echo $table; // output  "mywebarticles"   
  
$result = mysql\_query("UPDATE $table ...");   
}   
?>

[up](http://php.net/manual/vote-note.php?id=73890&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=73890&page=language.variables.scope&vote=down)

-1

[***mod*** ¶](http://php.net/manual/en/language.variables.scope.php#73890)

**7 years ago**

Can not access to global variables from destructor, if obj is not unseted at the end:  
  
<?php  
  
  class A  
   {  
     function \_\_destruct()  
      {  
        global $g\_Obj;  
        echo "<br>#step 2: ";  
        var\_dump($g\_Obj);  
      }  
  
     function start()  
      {  
        global $g\_Obj;  
        echo "<br>#step 1: ";  
        var\_dump($g\_Obj);  
      }  
   };  
  
  $g\_Obj = new A();        // start here  
  $g\_Obj->start();  
  $g\_Obj = NULL;        // !!! comment line and result will changed !!!  
  
?>  
  
Result, if line is not commented:  
  
#step 1: object(A)#1 (0) { }   
#step 2: object(A)#1 (0) { }  
  
Result, if line is commented:  
  
#step 1: object(A)#1 (0) { }   
#step 2: NULL

[up](http://php.net/manual/vote-note.php?id=68304&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=68304&page=language.variables.scope&vote=down)

-1

[***sami doesn't want spam at no-eff-eks com*** ¶](http://php.net/manual/en/language.variables.scope.php#68304)

**7 years ago**

PHP 5.1.4 doesn't seem to care about the static keyword. It doesn't let you use $this in a static method, but you can call class methods through an instance of the class using regular -> notation. You can also call instance methods as class methods through the class itself. The documentiation here is plain wrong.   
  
<?php   
class Foo {   
  public static function static\_fun()   
  {   
    return "This is a class method!\n";   
  }   
    
  public function not\_static\_fun()   
  {   
    return "This is an instance method!\n";   
  }   
}   
  
echo '<pre>';   
echo "From Foo:\n";   
echo Foo::static\_fun();   
echo Foo::not\_static\_fun();   
echo "\n";   
  
echo "From \$foo = new Foo():\n";   
$foo = new Foo();   
echo $foo->static\_fun();   
echo $foo->not\_static\_fun();   
echo '</pre>';   
?>   
  
You'll see the following output:   
  
From Foo:   
This is a class method!   
This is an instance method!   
  
From $foo = new Foo():   
This is a class method!   
This is an instance method!

[up](http://php.net/manual/vote-note.php?id=20407&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=20407&page=language.variables.scope&vote=down)

-1

[***huntsbox at pacbell dot net*** ¶](http://php.net/manual/en/language.variables.scope.php#20407)

**12 years ago**

Not sure of the implications of this but...   
You can create nested functions within functions but you must make sure they aren't defined twice, e.g.:   
  
<?php   
function norm($a, $b) {   
    static $first\_time = true;   
    if ($first\_time) {   
        function square($x) {   
            return $x \* $x;   
        }   
        $first\_time = false;   
    }   
    return sqrt(square($a) + square($b));   
}   
  
print square(5); // error, not defined yet   
print norm(5,4);   
print "<br>";   
print norm(3,2);   
print square(5); // OK   
?>   
  
If you don't include the if ($first\_time) you get an error saying you can't define square() twice.  Note that square is not local to the function it just appears there.  The last line successfully accesses square in the page scope.  This is not terribly useful, but interesting.

[up](http://php.net/manual/vote-note.php?id=78778&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=78778&page=language.variables.scope&vote=down)

-2

[***SID TRIVEDI*** ¶](http://php.net/manual/en/language.variables.scope.php#78778)

**6 years ago**

<?php  
/\*  
VARIABLE SCOPE : GLOBAL V/S STATIC  
  
If variable $count is defined global as under, instead of static, it does not work well as desired in repeated function calls.  
  
$count = 1; //if not defined STATIC, in each function call, it starts countig from one to 25.  
global $count;  
  
which gives folowing output:  
0123456789101112131415161718192021222324  
Total 24 numbers are printed.  
So far 26 function call(s) made.  
  
26272829303132333435363738394041424344454647484950  
Total 50 numbers are printed.  
So far 52 function call(s) made.  
\*/  
  
function print\_1to50()  
{  
//    $count = 1;  
//    global $count;  
    static $count=1; // Initial assigment of One to $count, static declarion holds the last(previous) value of variable $count in each next function calls.  
        $limit = $count+24;  
        while($count<=$limit)  
        {  
        echo "$count";  
        $count=$count+1;  
        }  
        $num\_count= $count-1;  
        echo "<br>\n". "Total $num\_count numbers are printed.<br>";  
  
        return; // return statement without parenthesis()or arguments denotes end of a function rather than returning any values to subsequent function call(s).  
} // end of while loop  
  
$count=0;  
print\_1to50();  
$count=$count+1;  
print "So far $count function call(s) made.<br><br>";  
  
print\_1to50();  
$count=$count+1;  
print "So far $count function call(s) made.<br>";  
/\*  
Which gives following output:  
12345678910111213141516171819202122232425  
Now I have printed 25 numbers.  
I have made 1 function call(s).  
26272829303132333435363738394041424344454647484950  
Now I have printed 50 numbers.  
I have made 2 function call(s).  
\*/  
  
?>

[up](http://php.net/manual/vote-note.php?id=52148&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=52148&page=language.variables.scope&vote=down)

-2

[***vdephily at bluemetrix dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#52148)

**9 years ago**

Be carefull about nested functions :  
<?php  
// won't work :  
function foo1()  
{  
  $who = "world";  
  function bar1()  
  {  
    global $who;  
    echo "Hello $who";  
  }  
}  
  
// will work :  
function foo2()  
{  
  $GLOBALS['who'] = "world";  
  function bar2()  
  {  
    global $who;  
    echo "Hello $who";  
  }  
}  
  
// also note, of course :  
function foo3()  
{  
  $GLOBALS['who'] = "world";  
  
  // won't work  
  echo "Hello $who";  
  
  // will work  
  global $who;  
  echo "Hello $who";  
}  
?>

[up](http://php.net/manual/vote-note.php?id=91982&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=91982&page=language.variables.scope&vote=down)

-1

[***emartin at sigb dot net*** ¶](http://php.net/manual/en/language.variables.scope.php#91982)

**5 years ago**

If you are used to include files which declare global variables, and if you now need to include these files in a function, you will see that those globals are declared in the function's scope and so they will be lost at the end of the function.  
  
You may use something like this to solve this problem:  
  
main\_file.php :  
<?php   
  
//Some innocent variables which exist before the problem  
$a = 42;  
$b = 33;  
$c = 56;  
  
function some\_function() {  
    //Some variables that we don't want out of the function  
    $saucisse = "saucisse";  
    $jambon = "jambon";  
      
    //Let's include another file  
    $evalt = "require\_once 'anothertest\_include.php';";  
    $before\_eval\_vars = get\_defined\_vars();  
    eval($evalt);  
  
    //Let's extract the variables that were defined AFTER the call to 'eval'  
    $function\_variable\_names = array("function\_variable\_names" => 0, "before\_eval\_vars" => 0, "created" => 0);  
    //We can generate a list of the newly created variables by substracting the list of the variables of the function and the list of the variables which existed before the call to the list of current variables at this point  
    $created = array\_diff\_key(get\_defined\_vars(), $GLOBALS, $function\_variable\_names, $before\_eval\_vars);  
    //Now we globalize them  
    foreach ($created as $created\_name => $on\_sen\_fiche)  
        global $$created\_name;  
    //And we affect them  
    extract($created);  
      
}  
  
some\_function();  
print\_r(get\_defined\_vars());  
  
?>  
  
included\_file.php :  
<?php  
  
//Some variables that we want in the global scope of main\_file.php  
$included\_var\_one = 123;  
$included\_var\_two = 465;  
$included\_var\_three = 789;  
  
?>

[up](http://php.net/manual/vote-note.php?id=20313&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=20313&page=language.variables.scope&vote=down)

-1

[***jochen\_burkhard at web dot de*** ¶](http://php.net/manual/en/language.variables.scope.php#20313)

**12 years ago**

Please don't forget:   
values of included (or required) file variables are NOT available in the local script if the included file resides on a remote server:   
  
remotefile.php:   
  
<?PHP   
$paramVal=10;   
?>   
  
localfile.php:   
  
<?PHP   
include "<http://example.com/remotefile.php>";   
echo "remote-value= $paramVal";   
?>   
  
Will not work (!!)

[up](http://php.net/manual/vote-note.php?id=113836&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=113836&page=language.variables.scope&vote=down)

-2

[***Ray.Paseur often uses Gmail*** ¶](http://php.net/manual/en/language.variables.scope.php#113836)

**7 months ago**

Variable "Visibility" in PHP Object Oriented Programming is documented here:  
<http://php.net/manual/en/language.oop5.visibility.php>

[up](http://php.net/manual/vote-note.php?id=111164&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=111164&page=language.variables.scope&vote=down)

-3

[***warbo*** ¶](http://php.net/manual/en/language.variables.scope.php#111164)

**1 year ago**

An alternative to static variables is to use an anonymous function. Here's an example with a simple counter which goes up on each call:  
  
<?php  
  
// Static version  
$staticCounter = function() {  
    // Make $a static, so its value is preserved  
    static $a = 0;  
    $a++;  
    return $a;  
};  
echo $staticCounter();  
echo $staticCounter();  
echo $staticCounter();  
// Output:  
// 1  
// 2  
// 3  
  
// Closure version  
$counterFactory = function () {  
    // We \*assign\* $a in this function  
    $a = 0;  
    return function() use (&$a) {  // Notice the &  
        // We \*use\* $a in this function  
        $a++;  
        return $a;  
    };  
};  
// The above is a 'counter factory'; when called it returns a counter  
$closureCounter1 = $counterFactory();  
echo $closureCounter1();  
echo $closureCounter1();  
echo $closureCounter1();  
// Output:  
// 1  
// 2  
// 3  
  
// If we call our 'factory' again, we get another counter, independent of the first  
$closureCounter2 = $counterFactory();  
echo $closureCounter1();  
echo $closureCounter2();  
echo $closureCounter1();  
echo $closureCounter2();  
// Output (notice that we're calling both counters)  
// 4  
// 1  
// 5  
// 2  
  
// We can do it with arguments too  
$makeCounterFrom = function($start) {  
    return function() use (&$start) {  
        $start++;  
        return $start;  
    };  
};  
$yearCounter = $makeCounterFrom(2012);  
echo $yearCounter();  
echo $yearCounter();  
// Output  
// 2013  
// 2014  
?>  
  
This use of closures may be familiar from Javascript's 'one-shot' functions "(function() {...})()". In PHP we can do this on 2 lines using a variable, or with 'call\_user\_func'.  
  
It's useful when a regular function isn't enough, but a whole class is overkill.

[up](http://php.net/manual/vote-note.php?id=106178&page=language.variables.scope&vote=up)

[down](http://php.net/manual/vote-note.php?id=106178&page=language.variables.scope&vote=down)

-7

[***sideshowAnthony at googlemail dot com*** ¶](http://php.net/manual/en/language.variables.scope.php#106178)

**2 years ago**

It can be nice to use static variables in class member functions.   
This avoids a 'class global' like $this->template.   
Also, I like the system of get and set using the same function.   
  
<?php   
  
class my\_page   
{   
    public function template($name=null)   
    {   
        static $template = 'templates/page.html';   
        if ($name) $template = "templates/{$name}.html";   
        else return $template;   
    }   
}   
  
$p = new my\_page;   
$p->template('product');   
include $p->template();   
  
?>

<?php  
  
  //You can even add more Dollar Signs  
  
  $Bar = "a";  
  $Foo = "Bar";  
  $World = "Foo";  
  $Hello = "World";  
  $a = "Hello";  
  
  $a; //Returns Hello  
  $$a; //Returns World  
  $$$a; //Returns Foo  
  $$$$a; //Returns Bar  
  $$$$$a; //Returns a  
  
  $$$$$$a; //Returns Hello  
  $$$$$$$a; //Returns World  
  
  //... and so on ...//  
  
?>

[up](http://php.net/manual/vote-note.php?id=31857&page=language.variables.variable&vote=up)

[down](http://php.net/manual/vote-note.php?id=31857&page=language.variables.variable&vote=down)

3

[***sir\_hmba AT yahoo DOT com*** ¶](http://php.net/manual/en/language.variables.variable.php#31857)

**11 years ago**

This is somewhat redundant, but I didn't see an example that combined dynamic reference of \*both\* object and attribute names.   
  
Here's the code:   
  
<?php   
class foo   
{   
    var $bar;   
    var $baz;   
  
    function foo()   
    {   
        $this->bar = 3;   
        $this->baz = 6;   
    }   
}   
  
$f = new foo();   
echo "f->bar=$f->bar  f->baz=$f->baz\n";   
  
$obj  = 'f';   
$attr = 'bar';   
$val  = $$obj->{$attr};   
  
echo "obj=$obj  attr=$attr  val=$val\n";   
?>   
  
And here's the output:   
  
f->bar=3  f->baz=6   
$obj=f  $attr=bar  $val=3

[up](http://php.net/manual/vote-note.php?id=25314&page=language.variables.variable&vote=up)

[down](http://php.net/manual/vote-note.php?id=25314&page=language.variables.variable&vote=down)

3

[***antony dot booth at nodomain dot here*** ¶](http://php.net/manual/en/language.variables.variable.php#25314)

**11 years ago**

You may think of using variable variables to dynamically generate variables from an array, by doing something similar to: -  
  
<?php  
foreach ($array as $key => $value)   
{  
  $$key= $value;  
}  
  
?>  
  
This however would be reinventing the wheel when you can simply use:   
  
<?php  
extract( $array, EXTR\_OVERWRITE);  
?>  
  
Note that this will overwrite the contents of variables that already exist.  
  
Extract has useful functionality to prevent this, or you may group the variables by using prefixes too, so you could use: -  
  
EXTR\_PREFIX\_ALL  
  
<?php  
$array =array("one" => "First Value",  
"two" => "2nd Value",  
"three" => "8"  
                );  
             
extract( $array, EXTR\_PREFIX\_ALL, "my\_prefix\_");  
     
?>  
  
This would create variables: -  
$my\_prefix\_one   
$my\_prefix\_two  
$my\_prefix\_three  
  
containing: -  
"First Value", "2nd Value" and "8" respectively

[up](http://php.net/manual/vote-note.php?id=24296&page=language.variables.variable&vote=up)

[down](http://php.net/manual/vote-note.php?id=24296&page=language.variables.variable&vote=down)

3

[***J. Dyer*** ¶](http://php.net/manual/en/language.variables.variable.php#24296)

**11 years ago**

Another use for this feature in PHP is dynamic parsing..    
  
Due to the rather odd structure of an input string I am currently parsing, I must have a reference for each particular object instantiation in the order which they were created.  In addition, because of the syntax of the input string, elements of the previous object creation are required for the current one.    
  
Normally, you won't need something this convolute.  In this example, I needed to load an array with dynamically named objects - (yes, this has some basic Object Oriented programming, please bare with me..)   
  
<?php   
   include("obj.class");   
  
   // this is only a skeletal example, of course.   
   $object\_array = array();   
  
   // assume the $input array has tokens for parsing.   
   foreach ($input\_array as $key=>$value){   
      // test to ensure the $value is what we need.   
         $obj = "obj".$key;   
         $$obj = new Obj($value, $other\_var);   
         Array\_Push($object\_array, $$obj);   
      // etc..   
   }   
  
?>   
  
Now, we can use basic array manipulation to get these objects out in the particular order we need, and the objects no longer are dependant on the previous ones.   
  
I haven't fully tested the implimentation of the objects.  The  scope of a variable-variable's object attributes (get all that?) is a little tough to crack.  Regardless, this is another example of the manner in which the var-vars can be used with precision where tedious, extra hard-coding is the only alternative.   
  
Then, we can easily pull everything back out again using a basic array function: foreach.   
  
<?php   
//...   
   foreach($array as $key=>$object){   
  
      echo $key." -- ".$object->print\_fcn()." <br/>\n";   
  
   } // end foreach     
  
?>   
  
Through this, we can pull a dynamically named object out of the array it was stored in without actually knowing its name.

[up](http://php.net/manual/vote-note.php?id=11585&page=language.variables.variable&vote=up)

[down](http://php.net/manual/vote-note.php?id=11585&page=language.variables.variable&vote=down)

3

[***bpotier at edreamers dot org*** ¶](http://php.net/manual/en/language.variables.variable.php#11585)

**13 years ago**

A good example of the use of variable variables name. Imagine that you want to modify at the same time a list of more than one record from a db table.   
1) You can easily create a dynamic form using PHP. Name your form elements using a static name and the record id   
ex: <input name="aninput<?php echo $recordid?>" which gives in the output something like <input name="aninput15">   
  
2)You need to provide to your form action/submit script the list of records ids via an array serialized and urlencoded via an hidden field (to decode and un serialize once in the submit script)   
  
3) In the script used to submit you form you can access the input value by using the variable ${'aninput'.$recordid} to dynamically create as many UPDATE query as you need   
  
[Editor Note: Simply use an array instead, for example: <input name="aninput[<?php echo $recordid?>]" And loop through that array. -Philip]

[up](http://php.net/manual/vote-note.php?id=75669&page=language.variables.variable&vote=up)

[down](http://php.net/manual/vote-note.php?id=75669&page=language.variables.variable&vote=down)

9

[***Sinured*** ¶](http://php.net/manual/en/language.variables.variable.php#75669)

**7 years ago**

One interesting thing I found out: You can concatenate variables and use spaces. Concatenating constants and function calls are also possible.   
  
<?php   
define('ONE', 1);   
function one() {   
    return 1;   
}   
$one = 1;   
  
${"foo$one"} = 'foo';   
echo $foo1; // foo   
${'foo' . ONE} = 'bar';   
echo $foo1; // bar   
${'foo' . one()} = 'baz';   
echo $foo1; // baz   
?>   
  
This syntax doesn't work for functions:   
  
<?php   
$foo = 'info';   
{"php$foo"}(); // Parse error   
  
// You'll have to do:   
$func = "php$foo";   
$func();   
?>   
  
Note: Don't leave out the quotes on strings inside the curly braces, PHP won't handle that graciously.

[up](http://php.net/manual/vote-note.php?id=98641&page=language.variables.variable&vote=up)

[down](http://php.net/manual/vote-note.php?id=98641&page=language.variables.variable&vote=down)

15

[***mason*** ¶](http://php.net/manual/en/language.variables.variable.php#98641)

**4 years ago**

PHP actually supports invoking a new instance of a class using a variable class name since at least version 5.2  
  
<?php  
class Foo {  
   public function hello() {  
      echo 'Hello world!';  
   }  
}  
$my\_foo = 'Foo';  
$a = new $my\_foo();  
$a->hello(); //prints 'Hello world!'  
?>  
  
Additionally, you can access static methods and properties using variable class names, but only since PHP 5.3  
  
<?php  
class Foo {  
   public static function hello() {  
      echo 'Hello world!';  
   }  
}  
$my\_foo = 'Foo';  
$my\_foo::hello(); //prints 'Hello world!'  
?>

[up](http://php.net/manual/vote-note.php?id=113497&page=language.variables.variable&vote=up)

[down](http://php.net/manual/vote-note.php?id=113497&page=language.variables.variable&vote=down)

3

[***Aycan Yat*** ¶](http://php.net/manual/en/language.variables.variable.php#113497)

**8 months ago**

Sometimes you might wish to modify value of an existing variable by its name. This is easily accomplishable with a combination of using "passing by reference" and "variable variables".  
  
$first\_var = 1;  
$second\_var = 2;  
$third\_var = 3;  
  
$which\_one = array\_rand('first', 'second', 'third');  
//Let's consider the result is "second".  
  
$modifier = $$which\_one;  //Now $modifier has value 2.  
$modifier++; //Now $modifier's value is 3.  
echo $second\_var; //Prints out 2  
  
//Consider we wish to modify the value of $second\_var  
$modifier = &$$which\_one;  //Simply passing by reference  
$modifier++; //Now value of $second\_var is 3 too.  
echo $second\_var; //Prints out 3  
  
It's that simple!

[up](http://php.net/manual/vote-note.php?id=50912&page=language.variables.variable&vote=up)

[down](http://php.net/manual/vote-note.php?id=50912&page=language.variables.variable&vote=down)

9

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.variable.php#50912)

**9 years ago**

It may be worth specifically noting, if variable names follow some kind of "template," they can be referenced like this:  
  
<?php  
// Given these variables ...  
$nameTypes    = array("first", "last", "company");  
$name\_first   = "John";  
$name\_last    = "Doe";  
$name\_company = "PHP.net";  
  
// Then this loop is ...  
foreach($nameTypes as $type)  
  print ${"name\_$type"} . "\n";  
  
// ... equivalent to this print statement.  
print "$name\_first\n$name\_last\n$name\_company\n";  
?>  
  
This is apparent from the notes others have left, but is not explicitly stated.

[up](http://php.net/manual/vote-note.php?id=24534&page=language.variables.variable&vote=up)

[down](http://php.net/manual/vote-note.php?id=24534&page=language.variables.variable&vote=down)

3

[***thien\_tmpNOSPAM at hotmail dot com*** ¶](http://php.net/manual/en/language.variables.variable.php#24534)

**11 years ago**

You can also use variable variables and the string concat operator to generate suffixed (or prefixed) variables based on a base name.   
  
For instance, if you wanted to dynamically generate this series of variables:   
  
base1\_suffix1   
base1\_suffix2   
base2\_suffix1   
base2\_suffix2   
base3\_suffix1   
base3\_suffix2   
  
You can do this:   
  
<?php   
$bases = array('base1', 'base2', 'base3');   
$suffixes = array('suffix1', suffix2);   
foreach($bases as $base) {   
    foreach($suffixes as $suffix) {   
        ${$base.$suffix} = "whatever";   
        #...etc   
    }   
}   
?>

From HTML 5.1 Draft:  
<http://www.w3.org/html/wg/drafts/html/master/forms.html#naming-form-controls:-the-name-attribute>  
  
The name content attribute gives the name of the form control, as used in form submission and in the form element's elements object. If the attribute is specified, its value must not be the empty string.  
Any non-empty value for name is allowed.  
  
So use the format like this <select multiple name="beer[]"> is still in the HTML 5 standard.

[up](http://php.net/manual/vote-note.php?id=43949&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=43949&page=language.variables.external&vote=down)

4

[***lennynyktyk at yahoo dot com*** ¶](http://php.net/manual/en/language.variables.external.php#43949)

**10 years ago**

When dealing with multiple select boxes and the name=some\_name[] so that PHP will understand that is needs to interpet the input as an array an not as a single value. If you want to access this in Javascript you should assign an id attribute to the select box as well as the name attribute. Then proceed to use the id attribute in Javascript to reference the select box and the name attribute to reference the select box in PHP.  
Example  
  
<select multiple id="select\_id" name="select\_name[]">  
....  
  
</select>  
  
<?PHP  
    echo $select\_name[0];  
?>  
  
<script language="javascript">  
  document.forms[0].select\_id.options[0].selected = true;  
</script>  
  
I hope you get the idea

[up](http://php.net/manual/vote-note.php?id=81080&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=81080&page=language.variables.external&vote=down)

3

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.external.php#81080)

**6 years ago**

The full list of field-name characters that PHP converts to \_ (underscore) is the following (not just dot):  
chr(32) ( ) (space)  
chr(46) (.) (dot)  
chr(91) ([) (open square bracket)  
chr(128) - chr(159) (various)  
  
PHP irreversibly modifies field names containing these characters in an attempt to maintain compatibility with the deprecated register\_globals feature.

[up](http://php.net/manual/vote-note.php?id=47188&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=47188&page=language.variables.external&vote=down)

3

[***mattij at nitro fi no at no dot no*** ¶](http://php.net/manual/en/language.variables.external.php#47188)

**9 years ago**

If you try to refer or pass HTML-form data which has arrays with javascript remember that you should point to that array like this  
  
<script type="text/javascript">  
    window.opener.document.forms[0]["to[where][we][point]"];  
</script>

[up](http://php.net/manual/vote-note.php?id=77344&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=77344&page=language.variables.external&vote=down)

4

[***vierubino dot r3m0oFdisB1T at gmail dot com*** ¶](http://php.net/manual/en/language.variables.external.php#77344)

**6 years ago**

When you are using checkboxes to submit multiple choices, there is no need to use the complex method further down the page where you assign a unique name to each checkbox.  
  
Instead, just name each checkbox as the same array, e.g.:  
  
<input type="checkbox" name="items[]" value="foo" />  
<input type="checkbox" name="items[]" value="bar" />  
<input type="checkbox" name="items[]" value="baz" />  
  
This way your $\_POST["items"] variable will return as an array containing all and only the checkboxes that were clicked on.

[up](http://php.net/manual/vote-note.php?id=52466&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=52466&page=language.variables.external&vote=down)

4

[***user: "someuser" at mai1server "ua.fm"*** ¶](http://php.net/manual/en/language.variables.external.php#52466)

**9 years ago**

Numerous string like:   
  
<?php   
if (isset($\_POST["var1"]))   
    $var1=$\_POST["var1"];   
else $var1='';   
//...   
if (isset($\_POST["varN"]))   
    $varN=$\_POST["varN"];   
else $varN='';   
?>   
  
Can be replaced with:   
  
<?php   
get\_superglobal\_vars\_from\_POST('var1','...','varN');   
  
function get\_superglobal\_vars\_from\_POST()   
{   
    $numargs = func\_num\_args();   
    $setargs = 0; // for counting set variables   
    for ($i=0; $i<$numargs; $i++)   
    {   
    $varname=func\_get\_arg($i);   
    if (!isset($\_POST[$varname]))   
        $result='';   
    else   
    {   
        $result=$\_POST[$varname];   
        $setargs++;   
    }      
    $GLOBALS["$varname"]=$result;   
    }   
    return $setargs; // who cares?   
}   
?>

[up](http://php.net/manual/vote-note.php?id=52423&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=52423&page=language.variables.external&vote=down)

2

[***tim at timpauly dot com*** ¶](http://php.net/manual/en/language.variables.external.php#52423)

**9 years ago**

This code module can be added to every form using require\_once().   
It will process any and all form data, prepending each variable with   
a unique identifier (so you know which method was used to get the data).   
  
My coding could be neater, but this sure makes processing forms much easier!   
  
<?php   
// -----------------------------------------------------------------   
// Basic Data PHP module. This module captures all GET, POST   
// and COOKIE data and processes it into variables.   
// Coded April, 2005 by Timothy J. Pauly   
// -----------------------------------------------------------------   
//   
// coo\_ is prepended to each cookie variable   
// get\_ is prepended to each GET variable   
// pos\_ is prepended to each POST variable   
// ses\_ is prepended to each SESSION variable   
// ser\_ is prepended to each SERVER variable   
  
session\_start(); // initialize session data   
$ArrayList = array("\_POST", "\_GET", "\_SESSION", "\_COOKIE", "\_SERVER"); // create an array of the autoglobal arrays   
// we want to process   
  
foreach($ArrayList as $gblArray) // process each array in the array list   
{   
   $prefx = strtolower(substr($gblArray,1,3))."\_"; // derive the prepend string   
// from the autoglobal type name   
   $tmpArray = $$gblArray;   
   $keys = array\_keys($tmpArray); // extract the keys from the array being processed   
   foreach($keys as $key) // process each key   
    {   
         
    $arcnt = count($tmpArray[$key]);   
      
    if ($arcnt > 1) // Break down passed arrays and   
// process each element seperately   
    {   
      $lcount = 0;   
      foreach ($tmpArray[$key] as $dval)   
        {   
           $prkey = $prefx.$key; // create a new key string   
// with the prepend string added   
           $prdata['$prkey'] = $dval; // this step could be eliminated   
           ${$prkey}[$lcount] = $prdata['$prkey']; //create new key and insert the data   
           $lcount++;   
        }   
        
        } else { // process passed single variables   
          
                $prkey = $prefx.$key; // create a new key string   
// with the prepend string added   
                $prdata['$prkey'] = $tmpArray[$key]; // insert the data from   
// the old array into the new one   
                $$prkey = $prdata['$prkey']; // create the newly named   
// (prepended) key pair using variable variables :-)   
                 
                  
               }   
    }   
}   
  
// -------------------------------------------------------------   
?>

[up](http://php.net/manual/vote-note.php?id=41025&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=41025&page=language.variables.external&vote=down)

1

[***arjini at mac dot com*** ¶](http://php.net/manual/en/language.variables.external.php#41025)

**10 years ago**

When dealing with form inputs named\_like\_this[5] and javascript, instead of trying to get PHP to do something fancy as mentioned below, just try this on the javascript side of things:  
  
<form name="myForm">  
  
<script>  
my\_fancy\_input\_name = 'array\_of\_things[1]';  
/\* now just refer to it like this in the dom tree   
  
document[myForm][my\_fancy\_input\_name].value  
  
etc\*/  
</script>  
  
<input type="text" name="array\_of\_things[1]" value="1"/>  
</form>  
  
No fancy PHP, in fact, you shouldn't need to change your PHP at all.

[up](http://php.net/manual/vote-note.php?id=51727&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=51727&page=language.variables.external&vote=down)

2

[***tmk-php at infeline dot org*** ¶](http://php.net/manual/en/language.variables.external.php#51727)

**9 years ago**

To handle forms with or without [] you can do something like this:   
  
<?php   
    function repairPost($data) {   
        // combine rawpost and $\_POST ($data) to rebuild broken arrays in $\_POST   
        $rawpost = "&".file\_get\_contents("php://input");   
        while(list($key,$value)= each($data)) {   
            $pos = preg\_match\_all("/&".$key."=([^&]\*)/i",$rawpost, $regs, PREG\_PATTERN\_ORDER);          
            if((!is\_array($value)) && ($pos > 1)) {   
                $qform[$key] = array();   
                for($i = 0; $i < $pos; $i++) {   
                    $qform[$key][$i] = urldecode($regs[1][$i]);   
                }   
            } else {   
                $qform[$key] = $value;   
            }   
        }   
        return $qform;   
    }   
  
    // --- MAIN   
  
    $\_POST = repairPost($\_POST);   
?>   
  
The function will check every field in the $\_POST with the raw post data and rebuild the arrays that got lost.

[up](http://php.net/manual/vote-note.php?id=105228&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=105228&page=language.variables.external&vote=down)

2

[***walf*** ¶](http://php.net/manual/en/language.variables.external.php#105228)

**2 years ago**

WARNING! replacement of spaces and dots does not occur in array keys.  
  
E.g. If you have  
<input name="a. b[x. y]" value="foo" />  
  
var\_dump($\_POST);  
gives  
array(1) {  
  ["a\_\_b"]=>  
  array(1) {  
    ["x. y"]=>  
    string(3) "foo"  
  }  
}

[up](http://php.net/manual/vote-note.php?id=29118&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=29118&page=language.variables.external&vote=down)

1

[***keli at kmdsz dot ro*** ¶](http://php.net/manual/en/language.variables.external.php#29118)

**11 years ago**

image type inputs apparently return their "value" argument from Mozilla, but not from IEXplorer... :(  
  
example:  
  
<input type="image" name="sb" value="first" src="first.jpg">  
  
using a mozilla will give you   
  $sb="first" AND $sb\_x, $sb\_y ... whereas from IE there's just no $sb. :(  
  
[this in short form, as I'm still using trackvars :) ]

[up](http://php.net/manual/vote-note.php?id=30019&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=30019&page=language.variables.external&vote=down)

2

[***Anonymous*** ¶](http://php.net/manual/en/language.variables.external.php#30019)

**11 years ago**

"...the dot (period, full stop) is not a valid character in a PHP variable name."  
  
That's not completely correct, consider this example:  
$GLOBALS['foo.bar'] = 'baz';  
echo ${'foo.bar'};  
This will output baz as expected.

[up](http://php.net/manual/vote-note.php?id=63295&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=63295&page=language.variables.external&vote=down)

1

[***ch1902uk at hotmail dot com*** ¶](http://php.net/manual/en/language.variables.external.php#63295)

**8 years ago**

Regarding image input buttons, above where it says:  
  
"When the user clicks somewhere on the image, the accompanying form will be transmitted to the server with two \*additional\* variables, sub\_x and sub\_y. These contain the coordinates of the user click within the image."   
  
This is the case with Firefox (and probably other standards browsers), however my experience with Internet Explorer is that when image inputs are clicked, they only submit the location of the click on the button and \*not\* the name of the input.  
  
So if you have a form to move/delete entries like this  
  
entry[]  [delete\_0] [up\_0] [down\_0]  
entry[]   [delete\_1] [up\_1] [down\_1]  
entry[]   [delete\_2] [up\_2] [down\_2]  
  
Then submitting the form in firefox will give you post variables such as   
  
<?php  
   $\_POST['delete\_2'];   // "Delete" - button value  
   $\_POST['delete\_2\_x'];   // 23 - x coord   
   $\_POST['delete\_2\_y'];   // 3 - y coord   
?>  
  
In IE you only get  
  
<?php  
   $\_POST['delete\_2\_x'];   // 23 - x coord   
   $\_POST['delete\_2\_y'];   // 3 - y coord   
?>  
  
So if you are checking for what button was clicked do something like this  
  
<?php  
   for ($i = 0; $i < count($\_POST['entry']); $i++)  
   {  
      if (isset($\_POST['delete\_' . $i . '\_x']))  
      {  
         // do delete  
      }  
   }  
?>

[up](http://php.net/manual/vote-note.php?id=50546&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=50546&page=language.variables.external&vote=down)

1

[***Murat TASARSU*** ¶](http://php.net/manual/en/language.variables.external.php#50546)

**9 years ago**

if you want your multiple select returned variable in comma seperated form you can use this. hope that helps. regards...  
  
$myvariable   
   Array ( [0] => one [1] => two [2] => three )   
turns into  
   one,two,three  
  
<?php  
$myvariable="";  
$myseperator="";  
foreach ( $\_POST["myvariable"] as $v) {  
if (!isset($nofirstcomma)) $nofirstcomma=0; else $myseperator=",";  
$myvariable = $myvariable.$myseperator.$v;  
}  
echo $myvariable;  
?>

[up](http://php.net/manual/vote-note.php?id=37887&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=37887&page=language.variables.external&vote=down)

1

[***darren at sullivan dot net*** ¶](http://php.net/manual/en/language.variables.external.php#37887)

**10 years ago**

This function is a simple solution for getting the array of selectes from a checkbox list or a dropdown list out of the Querry String. I took an example posted earlier and simplified it.   
  
<?php   
function multi\_post\_item($repeatedString) {   
    // Gets the specified array of multiple selects and/or   
    // checkboxes from the Query String   
    $ArrayOfItems = array();   
    $raw\_input\_items = split("&", $\_SERVER["QUERY\_STRING"]);   
    foreach ($raw\_input\_items as $input\_item) {   
        $itemPair = split("=", $input\_item);   
        if ($itemPair[0] == $repeatedString) {   
            $ArrayOfItems[] = $itemPair[1];   
        }   
    }   
    return $ArrayOfItems;   
}   
?>   
  
Use the name of the field as the agrument. Example:   
  
<?php   
$Order = $\_GET['Order'];   
$Name = $\_GET['Name'];   
$States = multi\_post\_item('States');   
$Products = multi\_post\_item('Products');   
?>   
  
Be sure to check for NULL if there are no selections or boxes checked.

[up](http://php.net/manual/vote-note.php?id=30866&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=30866&page=language.variables.external&vote=down)

1

[***un shift at yahoo dot com*** ¶](http://php.net/manual/en/language.variables.external.php#30866)

**11 years ago**

This function takes a recurring form item from php://input and loads it into an array - useful for javascript/dom incompatibility with form\_input\_item[] names for checkboxes, multiple selects, etc.  The fread maxes out at 100k on this one.  I guess a more portable option would be pulling in ini\_get('post\_max\_size') and converting it to an integer.   
  
<?php   
function multi\_post\_item($input\_item\_name) {   
     $array\_output = array();   
     $in\_handle = fopen("php://input", "r");   
     $raw\_input\_items = split("&", urldecode(fread($in\_handle, 100000)));   
     foreach ($raw\_input\_items as $input\_item) {   
            // split this item into name/value pair   
            $item = split("=", $input\_item);   
            // form item name   
            $item\_name = $item[0];   
            // form item value   
            $item\_value = $item[1];   
            if ($item\_name == $input\_item\_name) {   
                    $array\_output[] = $item\_value;   
            }   
     }   
     return $array\_output;   
}   
?>

[up](http://php.net/manual/vote-note.php?id=30485&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=30485&page=language.variables.external&vote=down)

1

[***vb at bertola dot eu dot org*** ¶](http://php.net/manual/en/language.variables.external.php#30485)

**11 years ago**

For what I understand, since PHP 4.3 it is possible to access the content of a POST request (or other methods as well) as an input stream named php://input, example:  
  
readfile("php://input");     
[to display it]  
  
or  
  
$fp = fopen("php://input", "r");      
[to open it and then do whatever you want]  
  
This is very useful to access the content of POST requests which actually have a content (and not just variable-value couples, which appear in $\_POST).  
  
This substitutes the old $HTTP\_RAW\_POST\_DATA variable available in some of the previous 4.x versions. It is available for other upload methods different from POST too, but it is not available for POSTs with multipart/form-data content type, since the file upload handler has already taken care of the content in that case.

[up](http://php.net/manual/vote-note.php?id=11873&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=11873&page=language.variables.external&vote=down)

1

[***yasuo\_ohgaki at hotmail dot com*** ¶](http://php.net/manual/en/language.variables.external.php#11873)

**13 years ago**

Important:  Pay attention to the following security concerns when handling user submitted  data :   
  
<http://www.php.net/manual/en/security.registerglobals.php>   
<http://www.php.net/manual/en/security.variables.php>

[up](http://php.net/manual/vote-note.php?id=49574&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=49574&page=language.variables.external&vote=down)

1

[***jlratwil at yahoo dot com*** ¶](http://php.net/manual/en/language.variables.external.php#49574)

**9 years ago**

To get multiple selected (with "multiple" ) lists in <select> tag, make sure that the "name" attribute is added to braces, like this:  
  
<select multiple="multiple" name="users[]">  
     <option value="foo">Foo</option>  
     <option value="bar">Bar</option>  
</select>  
  
When submitted to PHP file (assume that you have a complete form) it will return an array of strings. Otherwise, it will just return the last element of the <select> tag you selected.

[up](http://php.net/manual/vote-note.php?id=18737&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=18737&page=language.variables.external&vote=down)

1

[***a at b dot c dot de*** ¶](http://php.net/manual/en/language.variables.external.php#18737)

**12 years ago**

As far as whether or not "[]" in name attributes goes, The HTML4.01 specification only requires that it be a case-insensitive CDATA token, which can quite happily include "[]". Leading and trailing whitespace may be trimmed and shouldn't be used.  
  
It is the id= attribute which is restricted, to a case-sensitive NAME token (not to be confused with a name= attribute).

[up](http://php.net/manual/vote-note.php?id=34820&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=34820&page=language.variables.external&vote=down)

1

[***kevinrlat nospam dot ccs dot neu dot edu*** ¶](http://php.net/manual/en/language.variables.external.php#34820)

**10 years ago**

if you use an array of checkboxes to submit info to a database or what have you, be careful of the case when no boxes are checked.  for example:  
  
<form method="post">  
<input type="checkbox" name="checkstuff[]" value="0">  
<input type="checkbox" name="checkstuff[]" value="1">  
<input type="checkbox" name="checkstuff[]" value="2">  
  
. . .  
  
</form>  
  
if these are submitted and none are checked, the $\_POST['checkstuff'] variable will not contain an empty array, but a NULL value.  this bothered me when trying to implode() the values of my checkboxes to insert into a database, i got a warning saying the 2nd argument was the wrong type.    
  
hope this helps!  
-kevin

[up](http://php.net/manual/vote-note.php?id=114372&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=114372&page=language.variables.external&vote=down)

0

[***johnbendie at gmail dot com*** ¶](http://php.net/manual/en/language.variables.external.php#114372)

**4 months ago**

Please describe how the request is built and passed to the script on the server side. That will make things more clearer. Or provide a link detailing such process. Don't assume anything.

[up](http://php.net/manual/vote-note.php?id=94607&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=94607&page=language.variables.external&vote=down)

0

[***POSTer*** ¶](http://php.net/manual/en/language.variables.external.php#94607)

**4 years ago**

Here's a simple function to give you an uncorrupted version of $\_POST:  
  
<?php  
// Function to fix up PHP's messing up POST input containing dots, etc.  
function getRealPOST() {  
    $pairs = explode("&", file\_get\_contents("php://input"));  
    $vars = array();  
    foreach ($pairs as $pair) {  
        $nv = explode("=", $pair);  
        $name = urldecode($nv[0]);  
        $value = urldecode($nv[1]);  
        $vars[$name] = $value;  
    }  
    return $vars;  
}  
?>

[up](http://php.net/manual/vote-note.php?id=39270&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=39270&page=language.variables.external&vote=down)

0

[***jim at jamesdavis dot it*** ¶](http://php.net/manual/en/language.variables.external.php#39270)

**10 years ago**

How to pass a numerically indexed array.   
This is the part inside the form. Notice that the name is not 'english[$r]' which you would normally write, but 'english[]'. PHP adds the index when it receives the post and it starts at 0.   
  
<?php   
  
for ($r=0; $r <= count($english)-1; $r++){   
         echo "<TEXTAREA NAME='english[]'>".$english[$r]."</TEXTAREA>";          
            
}   
?>   
<?php   
  
And this will get it out at the other end   
function retrieve\_english(){   
    for ($r=0; $r <= count($\_POST['english'])-1; $r++){   
        echo $\_POST['english'][$r]."<BR>";   
    }   
}   
?>   
  
Keys are useful but so are numerical indices!   
Cheers everyone

[up](http://php.net/manual/vote-note.php?id=37185&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=37185&page=language.variables.external&vote=down)

0

[***soeren at hattel dot dk*** ¶](http://php.net/manual/en/language.variables.external.php#37185)

**10 years ago**

The decision in PHP to translate a query string like:  
  
a=2&a=3&a=4   
  
into one single variable a=4 is simply strupid!   
  
The "wonderful" hack allowing multiple values to be read only if one uses:  
  
a[]=2&a[]=3&a[]=4  
  
is - at first sight - a nice feature but soon become a pain in the a..!  
  
In ASP and ASPX the first situation is handled as:  
  
a=2,3,4  
  
which is better than the PHP behaviour but still bad (what if your variable values contain commas?).  
  
It seems to me that the proper behaviour would be:  
  
a=2&a=3&a=4  
  
automatically generates an array with all the variables inside. I know this would require proper error handling but evevy things does anyway!

[up](http://php.net/manual/vote-note.php?id=30257&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=30257&page=language.variables.external&vote=down)

0

[***mail at paulodeon dot com*** ¶](http://php.net/manual/en/language.variables.external.php#30257)

**11 years ago**

If you have form data that could be coming in via either GET or POST and register\_globals is off (as it should be) use the empty() function to find out where the data is coming from, as in the following example   
  
<?php   
if(empty($\_GET)) {   
        $clientfilter = $\_POST['clientfilter'];   
        $branchfilter = $\_POST['branchfilter'];   
}   
if(empty($\_POST)) {   
    $clientfilter = $\_GET['clientfilter'];   
    $branchfilter = $\_GET['branchfilter'];   
}   
?>

[up](http://php.net/manual/vote-note.php?id=21752&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=21752&page=language.variables.external&vote=down)

0

[***hjncom at hjncom dot net*** ¶](http://php.net/manual/en/language.variables.external.php#21752)

**12 years ago**

I think '[' and ']' are valid characters for name attributes.  
  
<http://www.w3.org/TR/html401/interact/forms.html#h-17.4>  
-> InputType of 'name' attribute is 'CDATA'(not 'NAME' type)  
  
<http://www.w3.org/TR/html401/types.html#h-6.2>  
-> about CDATA('name' attribute is not 'NAME' type!)  
...CDATA is a sequence of characters from the document character set and may include character entities...  
  
<http://www.w3.org/TR/html401/sgml/entities.html>  
--> about Character entity references in HTML 4  
([ - &#91, ] - &#93)

[up](http://php.net/manual/vote-note.php?id=21401&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=21401&page=language.variables.external&vote=down)

0

[***jesper at codecrew dot dk*** ¶](http://php.net/manual/en/language.variables.external.php#21401)

**12 years ago**

Just to help others with the same stupid problem i have:  
  
I you use a checkbox in your form, it will only return the value specified in value if it is checked.  
  
ex. <input type="checkbox" value="yes">  
  
in php code you then write  
  
$checkboxchecked = ($checkbox == "yes");  
  
I guess :)

[up](http://php.net/manual/vote-note.php?id=74775&page=language.variables.external&vote=up)

[down](http://php.net/manual/vote-note.php?id=74775&page=language.variables.external&vote=down)

-1

[***t.montg AT gmail DOT com*** ¶](http://php.net/manual/en/language.variables.external.php#74775)

**7 years ago**

For anyone else having trouble figuring out how to access values in a SELECT element from a POST or GET form, you can't set the "id" attribute to the same thing as your "name" attribute.  i.e. don't do this:  
  
<?php  
  //Not so good  
  <select multiple="multiple" id="selectElem" name="selectElem[]">  
     <option value="ham">Ham</option>  
     <option value="cheese">Cheese</option>  
     <option value="hamcheese">Ham and Cheese</option>  
  </select>  
?>  
  
If you do the above, the variable $\_POST['selectElem'] will not be set.  Instead, either change the id or name attribute so that they are dissimilar.  i.e. do this:  
  
<?php  
  //So good (notice the new "id" value)  
  <select multiple="multiple" id="selectElemId" name="selectElem[]">  
     <option value="ham">Ham</option>  
     <option value="cheese">Cheese</option>  
     <option value="hamcheese">Ham and Cheese</option>  
  </select>  
?>  
  
Then you can access the value(s) of the SELECT element through the array $\_POST['selectElem'][] or $\_GET['selectElem'][].  It took me quite some time to figure out the problem.

CONSTANTS and PHP Class Definitions  
  
Using "define('MY\_VAR', 'default value')" INSIDE a class definition does not work. You have to use the PHP keyword 'const' and initialize it with a scalar value -- boolean, int, float, or string (no array or other object types) -- 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;         WRONG - Works INSIDE of a class definition.  
//const MAX\_VALUE = 1.0;         WRONG - Works INSIDE 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](http://php.net/manual/vote-note.php?id=114762&page=language.constants&vote=up)

[down](http://php.net/manual/vote-note.php?id=114762&page=language.constants&vote=down)

3

[***php at webflips dot net*** ¶](http://php.net/manual/en/language.constants.php#114762)

**3 months ago**

It is perfectly valid to use a built-in PHP keyword as a constant name - as long as you the constant() function to retrieve it later:  
  
<?php  
define('echo', 'My constant value');  
  
echo constant('echo'); // outputs 'My constant value'  
?>

[up](http://php.net/manual/vote-note.php?id=19363&page=language.constants&vote=up)

[down](http://php.net/manual/vote-note.php?id=19363&page=language.constants&vote=down)

7

[***katana at katana-inc dot com*** ¶](http://php.net/manual/en/language.constants.php#19363)

**12 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](http://php.net/manual/vote-note.php?id=52008&page=language.constants&vote=up)

[down](http://php.net/manual/vote-note.php?id=52008&page=language.constants&vote=down)

5

[***storm*** ¶](http://php.net/manual/en/language.constants.php#52008)

**9 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.

[up](http://php.net/manual/vote-note.php?id=52133&page=language.constants&vote=up)

[down](http://php.net/manual/vote-note.php?id=52133&page=language.constants&vote=down)

3

[***hafenator2000 at yahoo dot com*** ¶](http://php.net/manual/en/language.constants.php#52133)

**9 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](http://php.net/manual/vote-note.php?id=35064&page=language.constants&vote=up)

[down](http://php.net/manual/vote-note.php?id=35064&page=language.constants&vote=down)

3

[***ewspencer at industrex dot com*** ¶](http://php.net/manual/en/language.constants.php#35064)

**10 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

Don't let the comparison between const (in the global context) and define() confuse you: while define() allows expressions as the value, const does not. In that sense it behaves exactly as const (in class context) does.  
  
<?php  
  
// this works  
/\*\*  
\* Path to the root of the application  
\*/  
define("PATH\_ROOT", dirname(\_\_FILE\_\_));  
  
// this does not  
/\*\*  
\* Path to configuration files  
\*/  
const PATH\_CONFIG = PATH\_ROOT . "/config";  
  
// this does  
/\*\*  
\* Path to configuration files - DEPRECATED, use PATH\_CONFIG  
\*/  
const PATH\_CONF = PATH\_CONFIG;  
  
?>

[up](http://php.net/manual/vote-note.php?id=114828&page=language.constants.syntax&vote=up)

[down](http://php.net/manual/vote-note.php?id=114828&page=language.constants.syntax&vote=down)

0

[***kuzawinski dot marcin at NOSPAM dot gmail dot com*** ¶](http://php.net/manual/en/language.constants.syntax.php#114828)

**3 months ago**

Actually, there is a way, to (re)declare FALSE (also: False, false, TRUE, NULL, etc.) constant, even if it is already declared:  
  
<?php  
  
var\_dump(constant('I\_DONT\_EXIST')); // warning "Couldn't find constant I\_DONT\_EXIST"  
var\_dump(constant('FALSE'));  //no warning, output: bool(false)  
  
define('FALSE', TRUE);  
  
var\_dump(constant('FALSE')); //output: bool(true)   
  
// but...  
  
var\_dump(constant('false')); // output: bool(false)  
var\_dump(FALSE); // output: bool(false);  
  
?>

[up](http://php.net/manual/vote-note.php?id=107327&page=language.constants.syntax&vote=up)

[down](http://php.net/manual/vote-note.php?id=107327&page=language.constants.syntax&vote=down)

-1

[***0gb dot us at 0gb dot us*** ¶](http://php.net/manual/en/language.constants.syntax.php#107327)

**2 years ago**

While most constants are only defined in one namespace, the case-insensitive true, false, and null constants are defined in ALL namespaces. So, this is not valid:  
  
<?php namespace false;  
const ENT\_QUOTES = 'My value';  
echo ENT\_QUOTES;//Outputs as expected: 'My value'  
  
const FALSE = 'Odd, eh?';//FATAL ERROR! ?>  
  
Fatal error: Cannot redeclare constant 'FALSE' in /Volumes/WebServer/0gb.us/test.php on line 5

[up](http://php.net/manual/vote-note.php?id=107164&page=language.constants.syntax&vote=up)

[down](http://php.net/manual/vote-note.php?id=107164&page=language.constants.syntax&vote=down)

-14

[***timucinbahsi at gmail dot com*** ¶](http://php.net/manual/en/language.constants.syntax.php#107164)

**2 years ago**

Constant names shouldn't include operators. Otherwise php doesn't take them as part of the constant name and tries to evaluate them:  
  
<?php  
define("SALARY-WORK",0.02); // set the proportion  
  
$salary=SALARY-WORK\*$work; // tries to subtract WORK times $work from SALARY  
?>

Another way to determine PHP\_INT\_MIN:   
  
<?php   
define('PHP\_INT\_MIN', ~PHP\_INT\_MAX);   
?>   
  
It should work always:   
  
MAX for 8bit-signed: 01111111   
MIN for 8bit-signed: 10000000   
  
In 32 bits:   
php -r"echo (int)base\_convert(str\_repeat('1', 31), 2, 10) - PHP\_INT\_MAX;"   
0   
<?php echo ~(int)base\_convert(str\_repeat('1', 31), 2, 10); ?>   
-2147483648

[up](http://php.net/manual/vote-note.php?id=112754&page=reserved.constants&vote=up)

[down](http://php.net/manual/vote-note.php?id=112754&page=reserved.constants&vote=down)

2

[***Anonymous*** ¶](http://php.net/manual/en/reserved.constants.php#112754)

**11 months ago**

If you think a negation is too complex for determining INT\_MIN, use an overflow:  
  
<?php (int)(PHP\_INT\_MAX + 1); ?>

[up](http://php.net/manual/vote-note.php?id=113172&page=reserved.constants&vote=up)

[down](http://php.net/manual/vote-note.php?id=113172&page=reserved.constants&vote=down)

2

[***rdcapasso*** ¶](http://php.net/manual/en/reserved.constants.php#113172)

**10 months ago**

Volker's getOS() function needs to have the order of cases changed in the switch statement since "darwin" contains "win", which means that both "windows" and "darwin" will return self::OS\_WIN. I've moved the 'dar' case above the 'win' case:  
  
<?php  
class System {  
  
    const OS\_UNKNOWN = 1;  
    const OS\_WIN = 2;  
    const OS\_LINUX = 3;  
    const OS\_OSX = 4;  
  
    /\*\*  
     \* @return int  
     \*/  
    static public function getOS() {  
        switch (true) {  
            case stristr(PHP\_OS, 'DAR'): return self::OS\_OSX;  
            case stristr(PHP\_OS, 'WIN'): return self::OS\_WIN;  
            case stristr(PHP\_OS, 'LINUX'): return self::OS\_LINUX;  
            default : return self::OS\_UNKNOWN;  
        }  
    }  
  
}  
?>

[up](http://php.net/manual/vote-note.php?id=113443&page=reserved.constants&vote=up)

[down](http://php.net/manual/vote-note.php?id=113443&page=reserved.constants&vote=down)

-2

[***espertalhao04 at hotmail dot com*** ¶](http://php.net/manual/en/reserved.constants.php#113443)

**9 months ago**

here is one solution to calculate PHP\_INT\_SIZE, PHP\_INT\_MAX and PHP\_INT\_MIN :  
  
<?php  
define('PHP\_INT\_SIZE',4+4\*!!(255<<63));  
define('PHP\_INT\_MIN', 255<<(PHP\_INT\_SIZE\*8)-1);  
define('PHP\_INT\_MAX',~PHP\_INT\_MIN);  
?>  
  
PHP\_INT\_MAX and PHP\_INT\_SIZE are not a must to define, if your system already has support for them.

[up](http://php.net/manual/vote-note.php?id=112550&page=reserved.constants&vote=up)

[down](http://php.net/manual/vote-note.php?id=112550&page=reserved.constants&vote=down)

-8

[***Volker*** ¶](http://php.net/manual/en/reserved.constants.php#112550)

**1 year ago**

simple check for current OS:  
  
<?php  
  
class System {  
  
    const OS\_UNKNWON = 1;  
    const OS\_WIN = 2;  
    const OS\_LINUX = 3;  
    const OS\_OSX = 4;  
  
    /\*\*  
     \* @return int  
     \*/  
    static public function getOS() {  
        switch (true) {  
            case stristr(PHP\_OS, 'WIN'): return self::OS\_WIN;  
            case stristr(PHP\_OS, 'DAR'): return self::OS\_OSX;  
            case stristr(PHP\_OS, 'LINUX'): return self::OS\_LINUX;  
            default : return self::OS\_UNKNWON;  
        }  
    }  
  
}  
  
?>

Note that even though PHP borrows large portions of its syntax from C, the ',' is treated quite differently. It's not possible to create combined expressions in PHP using the comma-operator that C has, except in for() loops.  
  
Example (parse error):  
  
<?php  
  
$a = 2, $b = 4;  
  
echo $a."\n";  
echo $b."\n";  
  
?>  
  
Example (works):  
<?php  
  
for ($a = 2, $b = 4; $a < 3; $a++)  
{  
  echo $a."\n";  
  echo $b."\n";  
}  
  
?>  
  
This is because PHP doesn't actually have a proper comma-operator, it's only supported as syntactic sugar in for() loop headers. In C, it would have been perfectly legitimate to have this:  
  
int f()  
{  
  int a, b;  
  a = 2, b = 4;  
  
  return a;  
}  
  
or even this:  
  
int g()  
{  
  int a, b;  
  a = (2, b = 4);  
  
  return a;  
}  
  
In f(), a would have been set to 2, and b would have been set to 4.  
In g(), (2, b = 4) would be a single expression which evaluates to 4, so both a and b would have been set to 4.

[up](http://php.net/manual/vote-note.php?id=21750&page=language.expressions&vote=up)

[down](http://php.net/manual/vote-note.php?id=21750&page=language.expressions&vote=down)

12

[***Mattias at mail dot ee*** ¶](http://php.net/manual/en/language.expressions.php#21750)

**12 years ago**

A note about the short-circuit behaviour of the boolean operators.  
  
1. if (func1() || func2())  
Now, if func1() returns true, func2() isn't run, since the expression  
will be true anyway.  
  
2. if (func1() && func2())  
Now, if func1() returns false, func2() isn't run, since the expression  
will be false anyway.  
  
The reason for this behaviour comes probably from the programming  
language C, on which PHP seems to be based on. There the  
short-circuiting can be a very useful tool. For example:  
  
int \* myarray = a\_func\_to\_set\_myarray(); // init the array  
if (myarray != NULL && myarray[0] != 4321) // check  
    myarray[0] = 1234;  
  
Now, the pointer myarray is checked for being not null, then the  
contents of the array is validated. This is important, because if  
you try to access an array whose address is invalid, the program  
will crash and die a horrible death. But thanks to the short  
circuiting, if myarray == NULL then myarray[0] won't be accessed,  
and the program will work fine.

[up](http://php.net/manual/vote-note.php?id=11883&page=language.expressions&vote=up)

[down](http://php.net/manual/vote-note.php?id=11883&page=language.expressions&vote=down)

7

[***yasuo\_ohgaki at hotmail dot com*** ¶](http://php.net/manual/en/language.expressions.php#11883)

**13 years ago**

Manual defines "expression is anything that has value", Therefore, parser will give error for following code.   
  
<?php   
($val) ? echo('true') : echo('false');   
Note: "? : " operator has this syntax  "expr ? expr : expr;"   
?>   
  
since echo does not have(return) value and ?: expects expression(value).   
  
However, if function/language constructs that have/return value, such as include(), parser compiles code.   
  
Note: User defined functions always have/return value without explicit return statement (returns NULL if there is no return statement). Therefore, user defined functions are always valid expressions.   
[It may be useful to have VOID as new type to prevent programmer to use function as RVALUE by mistake]   
  
For example,   
  
<?php   
($val) ? include('true.inc') : include('false.inc');   
?>   
  
is valid, since "include" returns value.   
  
The fact "echo" does not return value(="echo" is not a expression), is less obvious to me.   
  
Print() and Echo() is NOT identical since print() has/returns value and can be a valid expression.

[up](http://php.net/manual/vote-note.php?id=112682&page=language.expressions&vote=up)

[down](http://php.net/manual/vote-note.php?id=112682&page=language.expressions&vote=down)

6

[***chriswarbo at gmail dot com*** ¶](http://php.net/manual/en/language.expressions.php#112682)

**11 months ago**

Note that there is a difference between a function and a function call, and both  
are expressions. PHP has two kinds of function, "named functions" and "anonymous  
functions". Here's an example with both:  
  
<?php  
// A named function. Its name is "double".  
function double($x) {  
  return 2 \* $x;  
}  
  
// An anonymous function. It has no name, in the same way that the string  
// "hello" has no name. Since it is an expression, we can give it a temporary  
// name by assigning it to the variable $triple.  
$triple = function($x) {  
  return 3 \* $x;  
};  
?>  
  
We can "call" (or "run") both kinds of function. A "function call" is an  
expression with the value of whatever the function returns. For example:  
  
<?php  
// The easiest way to run a function is to put () after its name, containing its  
// arguments (if any)  
$my\_numbers = array(double(5), $triple(5));  
?>  
  
$my\_numbers is now an array containing 10 and 15, which are the return values of  
double and $triple when applied to the number 5.  
  
Importantly, if we \*don't\* call a function, ie. we don't put () after its name,  
then we still get expressions. For example:  
  
<?php  
$my\_functions = array('double', $triple);  
?>  
  
$my\_functions is now an array containing these two functions. Notice that named  
functions are more awkward than anonymous functions. PHP treats them differently  
because it didn't use to have anonymous functions, and the way named functions  
were implemented didn't work for anonymous functions when they were eventually  
added.  
  
This means that instead of using a named function literally, like we can with  
anonymous functions, we have to use a string containing its name instead. PHP  
makes sure that these strings will be treated as functions when it's  
appropriate. For example:  
  
<?php  
$temp      = 'double';  
$my\_number = $temp(5);  
?>  
  
$my\_number will be 10, since PHP has spotted that we're treating a string as if  
it were a function, so it has looked up that named function for us.  
  
Unfortunately PHP's parser is very quirky; rather than looking for generic  
patterns like "x(y)" and seeing if "x" is a function, it has lots of  
special-cases like "$x(y)". This makes code like "'double'(5)" invalid, so we  
have to do tricks like using temporary variables. There is another way around  
this restriction though, and that is to pass our functions to the  
"call\_user\_func" or "call\_user\_func\_array" functions when we want to call them.  
For example:  
  
<?php  
$my\_numbers = array(call\_user\_func('double', 5), call\_user\_func($triple, 5));  
?>  
  
$my\_numbers contains 10 and 15 because "call\_user\_func" called our functions for  
us. This is possible because the string 'double' and the anonymous function  
$triple are expressions. Note that we can even use this technique to call an  
anonymous function without ever giving it a name:  
  
<?php  
$my\_number = call\_user\_func(function($x) { return 4 \* $x; }, 5);  
?>  
  
$my\_number is now 20, since "call\_user\_func" called the anonymous function,  
which quadruples its argument, with the value 5.  
  
Passing functions around as expressions like this is very useful whenever we  
need to use a 'callback'. Great examples of this are array\_map and array\_reduce.

[up](http://php.net/manual/vote-note.php?id=77291&page=language.expressions&vote=up)

[down](http://php.net/manual/vote-note.php?id=77291&page=language.expressions&vote=down)

6

[***winks716*** ¶](http://php.net/manual/en/language.expressions.php#77291)

**6 years ago**

reply to egonfreeman at gmail dot com  
04-Apr-2007 07:45   
  
the second example u mentioned as follow:  
=====================================  
  
$n = 3;  
$n \* $n++  
  
from 3 \* 3 into 3 \* 4. Post- operations operate on a variable after it has been 'checked', but it doesn't necessarily state that it should happen AFTER an evaluation is over (on the contrary, as a matter of fact).  
  
===========================================  
  
everything works correctly but one sentence should be modified:  
  
"from 3 \* 3 into 3 \* 4"  should be "from 3 \* 3 into 4 \* 3"  
  
best regards~ :)