Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types −

* Local variables
* Function parameters
* Global variables
* Static variables.

Local Variables

A variable declared in a function is considered local; that is, it can be referenced solely in that function.

<?php

$x = 4;

function assignx () {

$x = 0;

print "\$x inside function is $x. <br />";

}

assignx();

print "\$x outside of function is $x. <br />";

?>

**Static variable**

?php  
function myTest() {  
    static $x = 0;  
    echo $x;  
    $x++;  
}  
  
myTest();  
echo "<br>";  
myTest();  
echo "<br>";  
myTest();  
?>

## *What is variable scope, anyway?*

The **scope** of a variable in PHP is the context in which the variable was created, and in which it can be accessed. Essentially, PHP has 2 scopes:

The global keyword is used to access a global variable from within a function.

**Global**

The variable is accessible from anywhere in the script

**Local**

The variable is only accessible from within the function (or method) that created it

Variable scope — and, in particular, local scope — make your code easier to manage. If all your variables are global, they can be read and changed from anywhere in your script. This can cause chaos in large scripts as many different parts of the script attempt to work with the same variable. By restricting a variable to local scope, you limit the amount of code that can access that variable, making your code more robust, more modular, and easier to debug.

Here's a simple example that shows how global and local variables work:

<?php

$globalName = "Zoe";

function sayHello() {

$localName = "Harry";

echo "Hello, $localName!<br>";

}

sayHello();

echo "The value of \$globalName is: '$globalName'<br>";

echo "The value of \$localName is: '$localName'<br>";

?>

This script displays the following:

Hello, Harry!

The value of $globalName is: 'Zoe'

The value of $localName is: ''

In this script, we create 2 variables:

* **$globalName** is a global variable since it's not created inside any function.
* **$localName** is a local variable, created and used inside the function sayHello().

After creating the global variable and the function, the script calls sayHello(), which displays 'Hello, Harry!'. The script then attempts to display the values of the 2 variables by using echo. Here's what happens:

* Since **$globalName** was created outside the function, it is accessible from anywhere in the script — including this point — so its value, 'Zoe', is displayed.
* **$localName**, on the other hand, is only accessible from within the sayHello() function. Since the echo statement is outside the function, PHP won't let the code access this local variable. Instead, PHP assumes that the code wants to create a new global variable called $localName, which it sets to the default value of an empty string. This is why the second call to echo displays the value of $localName as ''.

|  |  |
| --- | --- |
| **Sr.No** | **Name & Description** |
| 1 | **\_\_LINE\_\_**  The current line number of the file. |
| 2 | **\_\_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 whereas in older versions it contained relative path under some circumstances. |
| 3 | **\_\_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. |
| 4 | **\_\_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. |
| 5 | **\_\_METHOD\_\_**  The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive). |

## *Superglobals explained*

PHP provides a special set of predefined global arrays containing various useful nuggets of information. These arrays are known as ***superglobals***because they're accessible from anywhere in your script — including inside functions — and you don't need to declare them as global using the globalkeyword.

Here's a full list of the superglobals available in PHP, as of version 5.3:

**$GLOBALS**

Contains a list of all global variables in the script (excluding superglobals)

**$\_GET**

Holds a list of all form fields sent by the browser using the GET request

**$\_POST**

Holds a list of all form fields sent by the browser using the POST request

**$\_COOKIE**

Holds a list of all cookies sent by the browser

**$\_REQUEST**

Contains all the keys and values in the $\_GET, $\_POST and $\_COOKIE arrays combined

**$\_FILES**

Holds a list of any files uploaded by the browser

**$\_SESSION**

Allows you to store and retrieve persistent session variables for the current browser

**$\_SERVER**

Holds server environment info such as the filename of the running script, and the IP address of the browser.

**$\_ENV**

Contains a list of environment variables passed to PHP. These can include variables provided by the shell, as well as CGI variables.

For example, we can use the $\_GET superglobal to retrieve a value included in the query string of the script's URL, and display the value in the page:

<?php

$yourName = $\_GET['yourName'];

echo "Hello, $yourName!";

?>

If you run the above script using a URL along the lines of http://www.example.com/script.php?yourName=Fred then the script displays:

Hello, Fred!

**Warning:** In a real-world script, you should *never* send user input directly to the browser like this. It's a security risk. Always check or filter the input first.

The $GLOBALS superglobal is handy because it lets you access global variables within functions without needing to use the global keyword. For example:

<?php

$globalName = "Zoe";

function sayHello() {

echo "Hello, " . $GLOBALS['globalName'] . "!<br>";

}

sayHello(); // Displays "Hello, Zoe!"

?>