PHP - Variable Scope

In PHP, the scope of a variable is the context within which it is defined and accessible to the extent in which it is accessible. Generally, a simple sequential PHP script that doesn't have any loop or a function etc., has a single scope. Any variable declared inside the "<? php" and "?>" tag is available throughout the program from the point of definition onwards.

Based on the scope, a PHP variable can be any of these three types –

- Local Variables
- Global Variables
- Static Variables

A variable in a main script is also made available to any other script incorporated with **include** or **require** statements.

Example

In the following example, a "test.php" script is included in the main script.

main.php

```
<?php
   $var=100;
   include "test.php";
?>
```

test.php

```
<?php
   echo "value of \$var in test.php : " . $var;
?>
```

When the main script is executed, it will display the following **output** –

```
value of $var in test.php : 100
```

However, when the script has a user defined function, any variable inside has a local scope. As a result, a variable defined inside a function can't be accessed outside. Variables defined outside (above) the function have a global scope.

Example

Take a look at the following example –

```php
<?php
   $var=100;   // global variable
   function myfunction() {
      $var1="Hello";    // local variable
      echo "var=$var  var1=$var1" . PHP_EOL;
   }
   myfunction();
   echo "var=$var  var1=$var1" . PHP_EOL;
?>
```

It will produce the following **output** –

```
var=  var1=Hello
var=100  var1=

PHP Warning:  Undefined variable $var in /home/cg/root/64504/main.php on line 5
PHP Warning:  Undefined variable $var1 in /home/cg/root/64504/main.php on line 8
```

Note that a global variable is not automatically available within the local scope of a function. Also, the variable inside a function is not accessible outside.

The "global" Keyword

To enable access to a global variable inside local scope of a function, it should be explicitly done by using the "**global**" keyword.

Example

The PHP script is as follows –

```php
<?php
   $a=10;
   $b=20;
   echo "Global variables before function call: a = $a b = $b" . PHP_EOL;
   function myfunction() {
      global $a, $b;
      $c=($a+$b)/2;
      echo "inside function a = $a b = $b c = $c" . PHP_EOL;
      $a=$a+10;
   }
   myfunction();
   echo "Variables after function call: a = $a b = $b c = $c";
?>
```

It will produce the following **output** –

```
Global variables before function call: a = 10 b = 20
inside function a = 10 b = 20 c = 15
Variables after function call: a = 20 b = 20 c =
PHP Warning:  Undefined variable $c in /home/cg/root/48499/main.php on line 12
```

Global variables can now be processed inside the function. Moreover, any changes made to the global variables inside the function will be reflected in the global namespace.

$GLOBALS Array

PHP stores all the global variables in an associative array called **$GLOBALS**. The name and value of the variables form the key-value pair.

Example

In the following PHP script, $GLOBALS array is used to access global variables –

```php
<?php
   $a=10;
```

```php
   $b=20;
   echo "Global variables before function call: a = $a b = $b" . PHP_EOL;

   function myfunction() {
      $c=($GLOBALS['a']+$GLOBALS['b'])/2;
      echo "c = $c" . PHP_EOL;
      $GLOBALS['a']+=10;
   }
   myfunction();
   echo "Variables after function call: a = $a b = $b c = $c";
?>
```

It will produce the following **output** –

```
Global variables before function call: a = 10 b = 20
c = 15
PHP Warning:  Undefined variable $c in C:\xampp\htdocs\hello.php on line 12
Variables after function call: a = 20 b = 20 c =
```

Static Variable

A variable defined with **static** keyword is not initialized at every call to the function. Moreover, it retains its value of the previous call.

Example

Take a look at the following example –

</> Open Compiler

```php
<?php
   function myfunction() {
      static $x=0;
      echo "x = $x" . PHP_EOL;
      $x++;
   }
   for ($i=1; $i<=3; $i++) {
      echo "call to function :$i : ";
      myfunction();
```

```
    }
?>
```

It will produce the following **output** –

```
call to function :1 : x = 0
call to function :2 : x = 1
call to function :3 : x = 2
```