El olvidado mundo de las variables en PHP

Variables locales; variables globales; variables súper globales; variables estáticas; variables variables; variables constantes... ¿de verdad tienes la seguridad de conocerlas todas?

Escrito por: **Eugenia Bahit** (GLAMP Hacker & eXtreme Programmer)

Eugenia es **Arquitecta de Software**, docente e instructora de tecnologías **GLAMP** (GNU/Linux, Apache, MySQL, Python y PHP) y **Agile coach** (UTN) especializada en Scrum y **eXtreme Programming**. Miembro de la **Free Software Foundation**, **The Linux Foundation** y **Debian Hackers**. Creadora de python-printr, Europio Engine y colaboradora de Vim.

Webs:
Cursos de programación: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

Redes sociales:
Twitter / Identi.ca: **@eugeniabahit**

Frecuentemente me encuentro con que una gran cantidad de programadores se refiere al ámbito de las variables de manera equivocada, confundiendo a veces, ámbito con tipo. Me pareció una buena idea centrarme en este punto con la esperanza de dejar el tema más claro.

Sobre el ámbito de una variable

Para comenzar, lo primero que debemos dejar en *"blanco sobre negro"* es **a qué nos referimos realmente cuando hablamos del ámbito de una variable**. Básicamente, el ámbito al que pertenece una variable **se refiere a la disponibilidad de dicha variable** en el contexto de la aplicación.

Una variable puede estar disponible a nivel global o local. PHP, también incorpora el concepto de variable súper global. Veamos a qué se refiere cada una.

Variable global

Son variables accesibles desde cualquier parte de la aplicación, independientemente del tipo de estructura de control del que se trate. Las variables globales son aquellas que el

programador, define con la palabra `global` delante del nombre de la variable.

Si una variable global se define dentro de una función, ésta deberá ser ejecutada previamente para que la variable esté disponible a nivel global:

```
function foo() {
    global $bar;
    $bar = 75;
}

# Esto fallará:
print $bar;

# Sin embargo, al llamar a foo(), $bar estará disponible fuera de la función:
foo();
print $bar;
```

Lo mismo sucede, si la variable global es definida dentro de un método de clase:

```
class Foo {

    static function bar() {
          global $bar;
          $bar = 75;
    }

}

Foo::bar();
print $bar;
```

Sin embargo, cualquier variable definida fuera de una función o método de clase, estará disponible a nivel global sin necesidad de ser declarada como `global`.

```
/*
Archivo foo.php
   Tanto $foo como $i son variables globales.
   Los cambios realizados dentro del bucle while, también aplican a nivel global.
*/

$foo = 75;

$i = 0;
while($i < 5) {
    $foo--;
    $i++;
}

Archivo bar.php
require_once 'foo.php';
print $foo;  # Salida: 70
```

Pero, si se quisiera acceder a `$foo` o a `$i` desde una función o método de clase, previamente deberá ser llamada con `global`:

```
Archivo foo.php
$foo = 75;

$i = 0;
while($i < 5) {
    $foo--;
    $i++;
}


Archivo bar.php
require_once 'foo.php';

function bar() {
    # Esto fallará:
    print $foo;

    # Pero llamando a global funcionará:
    global $foo;
    print $foo;
}

bar();
# Salida: 70
```

Vale aclarar que una variable global definida por el programador, deberá ser incluida en el *script* para que esté disponible. Vale decir que si en el archivo `foo.php` defino una variable `$foo` a la que deseo acceder desde el archivo `bar.php`, en este último, debo incluir al archivo `foo.php`.

Variable local

Las variables locales son aquellas que se definen dentro de una función o método de clase, sin anteceder la palabra global. Este tipo de variables, solo serán accesibles desde la función o método que las ha declarado.

```
function foo() {
    $bar = 75;
}

# Esto fallará ya que $bar solo está disponible dentro de foo()
print $bar;


class Foo {

    static function bar() {
          $bar = 75;
    }

}

# Esto también fallará ya que $bar solo está disponible dentro de Foo::bar()
print $bar;
```

Los parámetros de una función o método de clase también son variables locales:

```
function foo($bar=75) {
    # Esto funciona:
    print $bar;
}


# Pero esto, fala:
print $bar;
```

> *Toda variable definida dentro de una función o método de clase, será de ámbito local si no es declarada como global*

Variables súper globales

Las variables súper globales son -a nivel ámbito- variables globales y se diferencian de éstas en que:

- No son definidas por el programador, sino que vienen incorporadas al lenguaje;
- No necesitan ser llamadas mediante `global`;

Entre las variables súper globales más usuales, nos podremos encontrar a:

```
array $_POST
array $_FILES
array $_GET
array $_SERVER
array $_COOKIE
array $_SESSION
array $http_response_header
array $argv
int $argc
```

Sobre el tipo de las variables

Cuando hablamos de "tipo de variables" es necesario aclarar que no estaos haciendo referencia al tipo de datos de la variable sino, al "modo" de la variable.

Existen 4 tipos básicos de variables:

1. **Variables:**

   las convencionales :)

2. **Variables estáticas:**

aquellas variables locales que conservan su valor tras la ejecución de la función o método que las declara.

3. **Variables variables:**

   aquellas cuyo nombre de variable se define en tiempo de ejecución.

4. **Constantes:**

   similares a las variables de ámbito global con la gran diferencia de que poseen un valor "constante" (no varía)

```
function foo() {
    $variable_local = 'bar';
    global $variable_global;
    $$variable_local = 'variable variable cuyo nombre es $bar';
}

const CONSTANTE = 0;
```

Variables estáticas

Una variable estática es de ámbito local pero se diferencia de una variable tradicional por el hecho de que conserva su valor tras la ejecución del ámbito.

Mientras que una variable local desaparece tras la ejecución del ámbito, provoca que su valor se restablezca en cada llamada:

```
function foo() {
    $bar = 1;
    $bar++;
    print $bar;
}

foo();
# Salida: 2

foo();
# Salida: 2
```

Sin embargo, al declararla como estática, conserva su valor:

```
function foo() {
    static $bar;
    $bar++;
    print $bar;
}
```

```
foo();
# Salida: 1

foo();
# Salida: 2

foo();
# Salida: 3

foo();
# Salida: 4
```

Como se puede apreciar, en cada llamada, la variable estática `$bar` ha conservado su último valor.

Variables variables

Las variables variables son una forma de definir variables -locales o globales- en tiempo de ejecución y resultan ideales en instrucciones iterativas. Son variables cuyo nombre, se define en tiempo de ejecución dejando disponible una nueva variable.

```
$nombre = 'foo';
$$nombre = 75;  # esta variable se llama $foo
print $foo;
# Salida: 75
```

Un uso frecuente de variables variables se da en estructuras iterativas. Un clásico ejemplo es la definición de variables para los datos enviados desde un formulario.

La forma típica de definición sería:

```
$nombre = $_POST['nombre'];
$apellido = $_POST['apellido'];
$email = $_POST['email'];
$cumpleanios = $_POST['cumpleanios'];
$dni = $_POST['dni'];
$nacionalidad = $_POST['nacionalida'];
```

Sin embargo, una forma más corta, se logra mediante el uso de variables variables:

```
foreach($_POST as $key=>$value) $$key = $value;
```

Lo anterior, dejará disponibles las siguientes variables:

```
$nombre          $apellido          $email       $cumpleanios
$dni             $nacionalidad
```

Constantes

Hasta antes de la versión 5.3 de PHP, una constante se definía con la función `define()`.

```
define('FOO', 1);
print FOO;  # Salida: 1

function bar() {
    print FOO;
}

bar();  # Salida: 1
```

En PHP 5.3 se incorpora la definición de constantes de ámbito global mediante **`const`** extendiendo así dicha instrucción que en la versión 5.2 solo se encontraba disponible en el ámbito de una clase.

```
const FOO = 1;
```

Mientras que una constante definida mediante `define()` podía recibir una expresión como valor:

```
$bar = 12;
define('FOO', $bar);
print FOO;  # Salida: 12

define('BAR', $bar * 2 / 3);
print BAR;  # Salida: 8
```

Una constante definida mediante const, solo puede recibir valores directos pero no expresiones:

```
# CORRECTO
const FOO = 1;
const BAR = FOO;

# INCORRECTO: Los siguientes casos, fallan:
$bar = 15;
const BAR2 = $bar;
const FOOBAR = 5 * 3 / 2;
```