Material PHP

Material extraído del manual en línea que se encuentra en el siguiente link:

https://www.php.net/manual/es/language.variables.php
https://www.php.net/manual/es/language.variables.basics.php
https://www.php.net/manual/es/language.variables.predefined.php
https://www.php.net/manual/es/language.variables.scope.php

Variables

Tabla de contenidos

Conceptos básicos

En PHP las variables se representan con un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable válido tiene que empezar con una letra o un carácter de subrayado (underscore), seguido de cualquier número de letras, números y caracteres de subrayado. Como expresión regular se podría expresar como: `'[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'`

Nota: Para los propósitos de este manual, una letra es a-z, A-Z, y los bytes del 127 al 255 (`0x7f-0xff`).

Nota: `$this` es una variable especial que no puede ser asignada.

Sugerencia

Vea también Guía de entorno de usuario para nombres.

Para más información sobre funciones relacionadas con variables, vea la Referencia de Funciones de Variables.

```php
<?php
$var = 'Roberto';
```

```
$Var = 'Juan';
echo "$var, $Var";      // imprime "Roberto, Juan"

$4site = 'aun no';      // inválido; comienza con un número
$_4site = 'aun no';     // válido; comienza con un carácter de subraya
do
$täyte = 'mansikka';    // válido; 'ä' es ASCII (Extendido) 228
?>
```

De forma predeterminada, las variables siempre se asignan por valor. Esto significa que cuando se asigna una expresión a una variable, el valor completo de la expresión original se copia en la variable de destino. Esto quiere decir que, por ejemplo, después de asignar el valor de una variable a otra, los cambios que se efectúen a una de esas variables no afectará a la otra. Para más información sobre este tipo de asignación, vea Expresiones.

PHP también ofrece otra forma de asignar valores a las variables: asignar por referencia. Esto significa que la nueva variable simplemente referencia (en otras palabras, "se convierte en un alias de" ó "apunta a") la variable original. Los cambios a la nueva variable afectan a la original, y viceversa.

Para asignar por referencia, simplemente se antepone un signo ampersand (&) al comienzo de la variable cuyo valor se está asignando (la variable fuente). Por ejemplo, el siguiente segmento de código produce la salida '`Mi nombre es Bob`' dos veces:

```
<?php
$foo = 'Bob';                 // Asigna el valor 'Bob' a $foo
$bar = &$foo;                 // Referenciar $foo vía $bar.
$bar = "Mi nombre es $bar";   // Modifica $bar...
echo $bar;
echo $foo;                    // $foo también se modifica.
?>
```

Algo importante a tener en cuenta es que sólo las variables con nombre pueden ser asignadas por referencia.

```
<?php
$foo = 25;
$bar = &$foo;       // Esta es una asignación válida.
$bar = &(24 * 7);   // Inválida; referencia una expresión sin nombre.

function test()
{
   return 25;
}

$bar = &test();     // Inválido.
?>
```

No es necesario inicializar variables en PHP, sin embargo, es una muy buena práctica. Las variables no inicializadas tienen un valor predeterminado de acuerdo a su tipo dependiendo del contexto en el que son usadas - las booleanas se asumen como **`false`**,

los enteros y flotantes como cero, las cadenas (p.ej. usadas en echo) se establecen como una cadena vacía y los arrays se convierten en un array vacío.

Ejemplo #1 Valores predeterminados en variables sin inicializar

```
<?php
// Una variable no definida Y no referenciada (sin contexto de uso); i
mprime NULL
var_dump($variable_indefinida);

// Uso booleano; imprime 'false' (Vea operadores ternarios para más in
formación sobre esta sintaxis)
echo($booleano_indefinido ? "true\n" : "false\n");

// Uso de una cadena; imprime 'string(3) "abc"'
$cadena_indefinida .= 'abc';
var_dump($cadena_indefinida);

// Uso de un entero; imprime 'int(25)'
$int_indefinido += 25; // 0 + 25 => 25
var_dump($int_indefinido);

// Uso de flotante/doble; imprime 'float(1.25)'
$flotante_indefinido += 1.25;
var_dump($flotante_indefinido);

// Uso de array; imprime array(1) {  [3]=>  string(3) "def" }
$array_indefinida[3] = "def"; // array() + array(3 => "def") => array(
3 => "def")
var_dump($array_indefinida);

// Uso de objetos; crea un nuevo objeto stdClass (vea http://www.php.n
et/manual/en/reserved.classes.php)
// Imprime: object(stdClass)#1 (1) {  ["foo"]=>  string(3) "bar" }
$objeto_indefinido->foo = 'bar';
var_dump($objeto_indefinido);
?>
```

Depender del valor predeterminado de una variable sin inicializar es problemático al incluir un archivo en otro que use el mismo nombre de variable. Un error de nivel E_NOTICE es emitido cuendo se trabaja con variables sin inicializar, con la excepción del caso en el que se anexan elementos a un array no inicializado. La construcción del lenguaje isset() puede ser usada para detectar si una variable ya ha sido inicializada.

Variables Predefinidas

PHP proporciona una gran cantidad de variables predefinidas a cualquier script que se ejecute. Muchas de éstas, sin embargo, no pueden ser completamente documentadas ya que dependen del servidor que esté corriendo, la versión y configuración de dicho servidor, y otros factores. Algunas de estas variables no estarán disponibles cuando se

ejecute PHP desde la línea de comandos. Para obtener una lista de estas variables, por favor vea la sección sobre Variables Predefinidas Reservadas.

PHP ofrece un conjunto adicional de arrays predefinidas que contienen variables del servidor web, el entorno y entradas del usuario. Estos nuevos arrays son un poco especiales porque son automáticamente globales. Por esta razón, son conocidas a menudo como "superglobales". Las superglobales se mencionan más abajo; sin embargo para una lista de sus contenidos y más información sobre variables predefinidas en PHP, por favor consulte la sección Variables predefinidas reservadas. Asimismo, podrá notar cómo las antiguas variables predefinidas (*$HTTP_*_VARS*) todavía existen.

Nota: **Variables variables**

Las superglobales no pueden ser usadas como variables variables al interior de funciones o métodos de clase.

Nota:

Aún cuando las superglobales y `HTTP_*_VARS` pueden existir al mismo tiempo; estas variables no son idénticas, así que modificar una no cambia la otra.

Si ciertas variables no son definidas en variables_order, los arrays de PHP predefinidos asociados a estas, estarán vacíos.

PHP proporciona una gran cantidad de variables predefinidas para todos los scripts. Las variables representan de todo, desde variables externas hasta variables de entorno incorporadas, desde los últimos mensajes de error hasta los últimos encabezados recuperados.

Tabla de contenidos

Ámbito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluídos y los requeridos. Por ejemplo:

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

Aquí, la variable *$a* estará disponible al interior del script incluido *b.inc*. Sin embargo, al interior de las funciones definidas por el usuario se introduce un ámbito local a la función. Cualquier variable usada dentro de una función está, por omisión, limitada al ámbito local de la función. Por ejemplo:

```
<?php
$a = 1; /* ámbito global */

function test()
{
    echo $a; /* referencia a una variable del ámbito local */
}

test();
?>
```

Este script no producirá salida, ya que la sentencia echo utiliza una versión local de la variable *$a*, a la que no se ha asignado ningún valor en su ámbito. Puede que usted note que hay una pequeña diferencia con el lenguaje C, en el que las variables globales están disponibles automáticamente dentro de la función a menos que sean expresamente sobreescritas por una definición local. Esto puede causar algunos problemas, ya que la gente puede cambiar variables globales inadvertidamente. En PHP, las variables globales deben ser declaradas globales dentro de la función si van a ser utilizadas dentro de dicha función.

La palabra clave `global`

En primer lugar, un ejemplo de uso de `global`:

Ejemplo #1 Uso de `global`

```
<?php
$a = 1;
$b = 2;

function Suma()
{
    global $a, $b;

    $b = $a + $b;
}
```

```
Suma();
echo $b;
?>
```

El script anterior producirá la salida `3`. Al declarar *$a* y *$b* globales dentro de la función, todas las referencias a tales variables se referirán a la versión global. No hay límite al número de variables globales que se pueden manipular dentro de una función.

Un segundo método para acceder a las variables desde un ámbito global es usando el array *$GLOBALS*. El ejemplo anterior se puede reescribir así:

Ejemplo #2 Uso de *$GLOBALS* en lugar de global

```
<?php
$a = 1;
$b = 2;

function Suma()
{
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}

Suma();
echo $b;
?>
```

El array *$GLOBALS* es un array asociativo con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento del array. *$GLOBALS* existe en cualquier ámbito, esto ocurre ya que *$GLOBALS* es una superglobal. Aquí hay un ejemplo que demuestra el poder de las superglobales:

Ejemplo #3 Ejemplo que demuestra las superglobales y el ámbito

```
<?php
function test_global()
{
    // La mayoría de variables predefinidas no son "super" y requieren
    // 'global' para estar disponibles al ámbito local de las funcione
s.
    global $HTTP_POST_VARS;

    echo $HTTP_POST_VARS['name'];

    // Las superglobales están disponibles en cualquier ámbito y no
    // requieren 'global'. Las superglobales están disponibles desde
    // PHP 4.1.0, y ahora HTTP_POST_VARS se considera obsoleta.
    echo $_POST['name'];
}
?>
```

Nota:

Utilizar una clave `global` fuera de una función no es un error. Esta puede ser utilizada aún si el fichero está incluido desde el interior de una función.

Uso de variables `static`

Otra característica importante del ámbito de las variables es la variable *estática*. Una variable estática existe sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito. Consideremos el siguiente ejemplo:

Ejemplo #4 Ejemplo que demuestra la necesidad de variables estáticas

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

Esta función tiene poca utilidad ya que cada vez que es llamada asigna a *$a* el valor `0` e imprime un `0`. La sentencia *$a*++, que incrementa la variable, no sirve para nada, ya que en cuanto la función finaliza, la variable *$a* desaparece. Para hacer una función útil para contar, que no pierda la pista del valor actual del conteo, la variable *$a* debe declararse como estática:

Ejemplo #5 Ejemplo del uso de variables estáticas

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

Ahora, *$a* se inicializa únicamente en la primera llamada a la función, y cada vez que la función `test()` es llamada, imprimirá el valor de *$a* y lo incrementa.

Las variables estáticas también proporcionan una forma de manejar funciones recursivas. Una función recursiva es la que se llama a sí misma. Se debe tener cuidado al escribir una función recursiva, ya que puede ocurrir que se llame a sí misma indefinidamente. Hay que asegurarse de implementar una forma adecuada de terminar la recursión. La siguiente función cuenta recursivamente hasta 10, usando la variable estática *$count* para saber cuándo parar:

Ejemplo #6 Variables estáticas con funciones recursivas

```
<?php
function test()
```

```
{
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        test();
    }
    $count--;
}
?>
```

Nota:

Las variables estáticas pueden ser declaradas como se ha visto en los ejemplos anteriores. Desde PHP 5.6 se pueden asignar valores a estas variables que sean el resultado de expresiones, aunque no se pueden usar funciones aquí, lo cual causaría un eror de análisis.

Ejemplo #7 Declaración de variables estáticas

```
<?php
function foo(){
    static $int = 0;          // correcto
    static $int = 1+2;        // correcto (a partir de PHP 5.6)
    static $int = sqrt(121);  // incorrecto (ya que es una función)

    $int++;
    echo $int;
}
?>
```

Nota:

Las declaraciones estáticas son resueltas en tiempo de compilación.

Referencias con variables `global` y `static`

El motor Zend 1, utilizado por PHP 4, implementa los modificadores static y global para variables en términos de referencias. Por ejemplo, una variable global verdadera importada dentro del ámbito de una función con `global` crea una referencia a la variable global. Esto puede ser causa de un comportamiento inesperado, tal y como podemos comprobar en el siguiente ejemplo:

```
<?php
function prueba_referencia_global() {
    global $obj;
    $obj = &new stdclass;
}

function prueba_no_referencia_global() {
    global $obj;
    $obj = new stdclass;
}
```

```php
prueba_referencia_global();
var_dump($obj);
prueba_no_referencia_global();
var_dump($obj);
?>
```

El resultado del ejemplo sería:

```
NULL
object(stdClass)(0) {
}
```

Un comportamiento similar se aplica a `static`. Las referencias no son almacenadas estáticamente.

```php
<?php
function &obtener_instancia_ref() {
    static $obj;

    echo 'Objeto estático: ';
    var_dump($obj);
    if (!isset($obj)) {
        // Asignar una referencia a la variable estática
        $obj = &new stdclass;
    }
    $obj->property++;
    return $obj;
}

function &obtener_instancia_no_ref() {
    static $obj;

    echo 'Objeto estático: ';
    var_dump($obj);
    if (!isset($obj)) {
        // Asignar el objeto a la variable estática
        $obj = new stdclass;
    }
    $obj->property++;
    return $obj;
}

$obj1 = obtener_instancia_ref();
$aun_obj1 = obtener_instancia_ref();
echo "\n";
$obj2 = obtener_instancia_no_ref();
$aun_obj2 = obtener_instancia_no_ref();
?>
```

El resultado del ejemplo sería:

```
Objeto estático: NULL
Objeto estático: NULL

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

Este ejemplo demuestra que al asignar una referencia a una variable estática, esta no es *recordada* cuando se invoca la funcion `&obtener_instancia_ref()` por segunda vez.