Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Tutorial PHP4 - Parte I (86094 lectures)

Per **Ricardo Galli Granada**, ***gallir*** (http://mnm.uib.es/gallir/)

Creado el 04/06/2001 22:58 modificado el 04/06/2001 22:58

Esta es la primera parte de una serie de artículos y tutoriales que estoy preparando para PHP4 y MySQL. Ahora presento la primera parte que se centra en introducir de forma breve las formalidades del lenguaje: sintaxis, expresiones, evaluación de expresiones, estructuras de control y funciones.

Curso de PHP4 Parte I

Ricardo Galli (gallir@uib.es)

PHP

PHP significa *Hypertext Preprocessor*, aunque originalmente significaba *Personal Home Page Tools*. Los ficheros PHP normalmente se denominan con la extensión `php`, `php3` o `phtml`.

El PHP es un lenguaje embebido en páginas HTML y que se ejecutan el servidor. Productos similares y propietarios son *Active Server Pages* (ASP) de Microsoft, *ColdFusion* de Allaire y *Java Server Pages* (JSP) de Sun.

PHP es fácil de aprender comparado con otros mecanismos para obtener la misma funcionalidad. A diferencia de JSP o CGI basados en C, PHP no requiere un conocimiento exhaustivo del lenguaje de programación. A diferencia de Perl, PHP tiene una sintaxis muy fácil de comprender y a diferencia de ASP, no requiere conocer más de un lenguaje de programación o de la instalación de módulos externos o comerciales para realizar tareas más complicadas no previstas en el lenguaje más usado (Visual Basic Script).

La mayoría de las funciones más útiles están predefinidas:

- Acceso a bases de datos: ODBC, Oracle, Postgres, SQL Server, MySQL, Informix, Interbase, SyBase, mSQL, dBase
- Conectividad: HTTP, FTP, COM, YP/NIS, SNMP, Sockets, CORBA, LDAP
- Servicios Correo y Noticias: POP, IMAP, SMTP, NNTP
- Textos y Gráficos: XML, HTML, PDF, GD, Flash
- Funciones Matemáticas.
- POSIX: semáforos, memoria compartida, acceso a ficheros, expresiones regulares, cronómetros
- Comercio Electrónico: Cybercash, Verisign
- Formularios.
- Encriptación y Compresión: MD5, Gzip, Bzip2, OpenSSL

Las instrucciones PHP están embebidas en HTML. Una página PHP es una página normal HTML que con unas marcas especiales le indican al servidor que deben interpretarse. Por ejemplo (los ejemplos de estas clases están en http://mnm.uib.es/~gallir/php[1]/:

ejemplo1.phtml

```
<html>
<head>
```

```
<title>Saludos</title>
</head>
<body>
<p>Hola,
<?
/* Ahora pasamos a modo PHP */
$nombre = "Ricardo";
$apellido = "Galli";
echo "soy $nombre $apellido";
?>
</p>
</body>
</html>
```

Cuando un cliente solicita esta página, el servidor web la procesa en forma secuencial desde el principio al final buscando secciones PHP limitadas por <? y ?>. En caso de encontrarlas, las compila y ejecuta. Si todo se ejecuta de forma normal, producirá la siguiente página HTMLs:

```
<HTML>

<HEAD>

<title>Saludos</title>

</head>

<body>

<p>Hola,

soy Ricardo Galli</p>

</BODY>

</HTML>
```

El resultado es idéntico a si hubiese sido escrito el texto manualmente. Esto tiene algunas consecuencias:

- PHP puede ser agregado rápidamente al código HTML producido por editores HTML interactivos.
- PHP facilita la interacción entre diseñadores y programadores.
- No se necesita re-escribir cada línea de código HTML en un lenguaje de programación.
- PHP reduce costes y aumenta la eficiencia.

Agregar PHP a HTML

PHP es totalmente compatible con HTML y no tomará en cuenta la inclusión de applets, Javascript, etc., simplemente los ignorará. Se puede usar cualquier método para generar HTML y luego se puede agregar PHP en él.

Para indicar las secciones PHP se deben usar etiquetas especiales, este proceso es llamado "escape del HTML". Las etiquetas válidas son:

1. Etiquetas canónicas PHP: `<?php ?>`
2. Etiquetas cortas (tipo SGML): `<? ?>`
3. Estilo ASP: `<% %>` (se debe verificar que esté habilitada esta opción en el php.ini
4. Etiquetas estilo HTML: `<SCRIPT LANGUAGE="PHP"> </SCRIPT>`

Hola Mundo

ejemplo2.phtml

```
<HTML>
<HEAD>
<TITLE>El primer programa en PHP</TITLE>
</HEAD>
<BODY>
<? print ("Hola mundo cruel");?>
</BODY>
</HTML>
```

Si le aburre mucho probar este ejemplo, cambie la función `print` por `phpinfo()` y analice los resultados obtenidos

Entrar y Salir del Modo PHP

En cualquier momento se puede entrar y salir del modo PHP. Todo lo que esté entre las etiquetas de escape es considerado PHP, todo lo que esté afuera es ignorados, no hay término medio. Por ejemplo es válido:

ejemplo3.phtml

```
<?php $id=1; ?>
<FORM METHOD="POST" ACTION="ejemplo3.phtml">
<P>Nombre:
<INPUT TYPE="TEXT" NAME="nombre" SIZE=20>
<P>Id:
<INPUT TYPE="TEXT" NAME="id" VALUE="<? echo $id ?>">
<INPUT TYPE="SUBMIT" VALUE="Enviar">
</FORM>
```

Inclusión de ficheros PHP

Otra forma de agregar código PHP al HTML es poniendo el código PHP en otro fichero e invocarlo mediante la función `include`. Por ejemplo, un fichero llamado `ejemplo4.inc` contiene solamente el siguiente código:

ejemplo4.inc

```
<? $novia = "Mafalda";
print $novia; ?>
```

ejemplo4.phtml

```
<HTML>
<HEAD>
<TITLE>Una carta sincera</TITLE>
</HEAD>
<BODY>
<P>Tienes que creerme si te digo que no puede vivir sin ti querida
<? include("ejemplo4.inc"); ?>.
El nombre "<? include("ejemplo4.inc"); ?>" resuena en mis oídos.
</BODY>
</HTML>
```

Hay que tener cuidado, el texto que se incluye del fichero `ejemplo4.inc` **no** está en modo PHP, sino que está en modo HTML. Esta es la razón que tengamos que volver a poner las etiquetas PHP para pasar a modo PHP nuevamente dentro del fichero incluido.

También se suele usar la función `require` en vez de `include`. Aunque `include` es más flexible, `require` es más rápido. `Include` también permite la inclusión de HTML debido a que se interpreta en modo HTML.

Sintaxis, Variables y Salida

PHP es bastante flexible, más que intentar ser estricto y forzar una disciplina en la programación, enfatiza conveniencia para el programador más que la corrección. El PHP tiene un conjunto mínimo de reglas que hay que seguir, caso contrario podremos ver los mensajes de error tipo `"Parser error in line XXX"`.

Sintaxis

La sintaxis del PHP es similar a la del lenguaje C, o sea, muy sencilla. Si no sabe como escribir una instrucción, pruebe primero como lo haría en C, y si no funciona acuda al manual.

PHP ignora los espacios en blanco

```
$valor=2+2;
```

es equivalente a

```
$valor      <TAB>=    2 +   2;
```

o a

```
$valor =

2 +

2;
```

PHP es a veces sensitivo a mayúsculas-minúsculas

En algunos casos PHP no es sensitivo a las mayúsculas o minúsculas, como en el caso de nombres de función o construcciones del lenguaje (`if, then. else, while, for`, etc.), pero para otros sí que lo es, por ejemplo para nombres de variables.

```
<?

      $total = 100;

      print "El total es $total <br>";

      print "El total es $Total <br>";

?>
```

La salida del programa anterior será:

El total es 100

El total es

Las sentencias son expresiones que terminan en punto y coma

La típica sentencia en PHP es una asignación:

```
$saludos = "Bienvenidos a PHP";
```

Los bloques de construcción más pequeños son las **palabras indivisibles**, tal como números (`2.718281`), cadenas (`"uno, dos, tres"`), variables (`$saludos`), constantes (`TRUE, FALSE`) y las construcciones que definen el lenguaje (`if, then `). Estas palabras se separan de las demás por espacios en blanco o caracteres especiales que hacen de separador, como paréntesis, operadores, llaves, etc.

Los siguientes elementos en complejidad son las **expresiones**, que son una combinación de palabras que tienen un **valor**. Las expresiones más simples están formadas por una sola palabra, como un número o una variable. Expresiones simples pueden ser combinadas con operadores para formar expresiones más complejas, por ejemplo `$variable + 2 * 3`.

Evaluación de expresiones

Cada vez que el intérprete encuentra una expresión, la expresión es inmediatamente evaluada. Esto significa que el PHP primera evalúa los elementos más pequeños y luego las combina y obtiene su resultado.

Por ejemplo en

```
$total = 2 * 3 + 4 * 5 + 6;
```

los pasos imaginarios de evaluación serán:

```
= 6 + 4 * 5 + 6

= 6 + 20 + 6

= 26 + 6

= 32
```

Finalmente el valor 32 es asignado a la variable `$total`.

Asignaciones

El tipo más común de expresión en la asignación, donde el resultado de una expresión es almacenado en una variable. La forma es el nombre de la variable, que comienza siempre con el símbolo $, seguido de un símbolo igual y a continuación la expresión a evaluar.

```
$ocho = 2 + 2 + 2 +2;
```

Una cosa importante a recordar es que las asignaciones son expresiones y por la tanto también tienen un resultado, el mismo que se asigna a la variable:

```
$doce = 4 + ($ocho = 2 + 2+ 2+ 2);
```

Expresiones y Sentencias

Normalmente hay dos razones para usar una expresión en PHP:

1. por su *valor*,
2. por su *efecto secundario*.

El *valor de una expresión* es aquello que se pasa para la evaluación de expresiones más complejas, los *efectos secundarioes* es aquello que ocurre como resultado de una evaluación. Los casos más típicos de efectos son la asignación de resultados a una variable, imprimir algo en la pantalla del usuario o cambiando valores en una base de

datos.

Aunque las sentencias son expresiones, no están incluidas en expresiones más complejas, lo que significa que la única razón para usar una sentencia es el efecto secundario.

```
print("Hola"); // el efecto secundario es imprimir en la pantalla

2 + 2 + 2; // no tiene sentido

$seis = 2 + 2+ 2; // el efecto secundario es la asignación
```

Construcción de Bloques

Aunque las sentencias no pueden ser combinadas como las expresiones, se puede poner una serie de sentencias en cualquier permitido agrupándolas con "{" y "}".

Por ejemplo, la construcción if está formada por una verificación (entre paréntesis) seguido de una sentencia que es ejecutada si la condición es verdadera.

```
if ( 3 == 2+1)

      print ("Si, es tres!!!<br>");
```

Si queremos que se ejecuten varias sentencias en vez de una sola, lo que hacemos es agrupar aquellas sentencias:

```
if ( 3 == 2+1) {

      print ("Si, es tres!!!<br>");

      print ("y no era cuatro.<p>");

}
```

El sangrado, aunque ignorado por el intérprete del PHP, **es muy importante** para la comprensión de los programas.

Comentarios

Los comentarios son porciones del programa que se ponen sólo para facilitar la comprensión de lectores, lo primero que hace el intérprete de PHP es quitar todos los comentarios del programa. La forma de los comentarios en PHP están heredados de varios lenguajes de programación muy usados en entornos UNIX.

Comentarios Tipo C

```
/* Este es un comentario

* similar al del lenguaje C */
```

Hay que tener en cuenta que los comentarios de este tipo no pueden estar anidados:

```
/* Este comentario /* dará unos

 errores horribles en la última */ palabra */
```

Comentarios de una línea: // y #

Estos tipos de comentarios están heredados del C++, Java, Shell y Perl.

```
// esta es una línea

// esta es la segunda línea
```

```
# esto también es un comentario

# y también este último.
```

Variables

La forma principal de almacenar valores en el medio de un programa son las variables. Las cosas más importantes a recordar son:

- Todas las variables en PHP comienzan con el símbolo dólar $ (heredado del Shell y Perl)
- El valor de una variable es igual al valor más recientemente asignado.
- Las variables son asignadas con el operado '=', con la variable a la izquierda del operador y la expresión a evaluar a la derecha.
- Las variables no necesitan ser declaradas antes de ser usadas.
- Las variables no tienen un valor intrínseco, sino que toman el tipo del último valor asignado.
- Las variables que se usan antes de ser asignadas tienen un valor por defecto.

Verificación de asignación previa

Se puede verificar que una variable haya sido previamente asignada con la función `IsSet()`:

```
if(IsSet($a))

print ("la variable tiene el valor $a");

else

print ("la variable a no ha sido asignada");
```

Ámbito de las variables

El ámbito (*scope*) es el término técnico para definir el comportamiento de un nombre, función o variable, dentro de un programa. Puede darse el caso que el nombre tenga el mismo comportamiento y significado en todo el programa (global) o que el mismo nombre se comporte de manera diferente en distintas partes del programa. En este último caso decimos que el ámbito de esas variables es local.

En PHP, cualquier variable que no esté dentro de una función tiene ámbito global y su valor se encuentra disponible en toda la extensión de la ejecución del programa. En otras palabras, si se asigna un valor a una variable al principio del programa, el nombre de la variable tiene el mismo significado para el resto del programa y tendrá siempre el mismo valor.

La asignación de un valor a una variable no afectará a las variables con el mismo nombre en la ejecución del PHP de otras páginas ni tampoco a la ejecución del mismo programa PHP cuando es invocado por conexiones diferentes.

La pregunta de si una variable persiste a través de las diferentes etiquetas PHP, es decir si se puede salir del modo PHP, volver a entrar luego y usar las mismas variables con los mismos valores de una asignación previa. Si, no hay problemas.

ejemplo5.phtml

```
<html>
<head>
<?  $nombre = "Ricardo"; ?>
<title>Saludos</title>
</head>
<body>
<p>
<? print ("Hola, soy $nombre"); ?>
</p>
```

```
</body>
</html>
```

Salida

La mayoría de las construcciones del PHP se ejecutan silenciosamente, es decir no producen ninguna salida hacia la pantalla del usuario. Si se desea enviar texto al navegador del usuario se deben usar las funciones de salida.

Echo y print

Las dos construcciones básicas para generar salida son `echo` y `print`. Aunque al principio su sintaxis puede parecer un poco confusa (como en los ejemplos vistos hasta ahora), esto se debe a que son construcciones básicas del lenguaje y no funciones, por lo tanto se **puede o no usar paréntesis**.

```
echo "Hola mundo";

echo ("Hola Mundo");

print "Hola mundo";

print ("Hola mundo");

print (3.1415926);
```

Variables y Cadenas

Se pueden usar `echo` y `printf` para imprimir de forma combinada cadenas y variables.

ejemplo6.phtml

```
<html>
<head>
<title>Animales</title>
</head>
<body>
<?
$animal = "cabra";
$lugar = "montañas";
$patas = 4;
print ("La $animal vive en las $lugar y tiene cuatro patas<br>");
?>
</body>
</html>
```

Comillas simples vs. comillas dobles

Es uso de las comillas simples o dobles cambia el comportamiento. Si, como en el ejemplo anterior, se usan comillas dobles, el PHP interpolará (cambiará) los nombres de variables que encuentre por su valor. En cambio si se usan comillas simples, el PHP lo tomará con una cadena que debe imprimirse tal cual han sido escritas.

ejemplo7.phtml

```
<html>
<head>
<title>Animales</title>
</head>
<body>
<?
$animal = "cabra";
```

```
$lugar = "montañas";
$patas = 4;
print ('La $animal vive en las $lugar y tiene cuatro patas<br>');
?>
</body>
</html>
```

Lo que producirá el siguiente resultado:

```
La $animal vive en las $lugar y tiene cuatro patas
```

Printf

Es posible generar una salida con formatos más complejos mediante el uso de la función `printf`, con una sintaxis similar al lenguaje C o Perl.

ejemplo8.phtml

```
<html>
<head>
<title>Animales</title>
</head>
<body>
<?
$animal = "cabra";
$lugar = "montañas";
$patas = 4;
printf( "El %s vive en las %s y tiene %d patas<br>", $animal, $lugar, $patas);
?>
</body>
</html>
```

Tipos

Todos los lenguajes de programación tienen un sistema que especifican los diferentes tipos de datos que pueden aparecer en los programas. Normalmente los diferentes tipos corresponden a la forma en que representan los valores en la memoria mediante una serie de bits. El sistema de tipos en PHP es extremadamente sencillo y flexible lo que facilita la tarea de los programadores.

Los tipos básicos de PHP son enteros (`integers`), flotantes (`doubles`), lógicos (`Booleans`), cadenas (`strings`), vectores (`arrays`) y objetos.

- *Integers* son números enteros, sin punto decimal, como `133`.
- *Doubles* son números de punto flotante, como `3.1416` o `0.001` o `41.0`.
- *Booleans* son valores lógicos y sólo permiten `TRUE` o `FALSE`.
- *Strings* son secuencias de caracteres como `"El PHP 4 es muy simple"` o `'PHP no es complicado'`.
- *Arrays* son una colección de datos indexados por alguna clave.

Strings

```
$str1 = "Una cadena con comillas dobles";

$str2 = 'Una cadena con comillas simples';

$str0 = "" //Cadena vacía
```

Comillas simples

A excepción de unos pocos casos, las cadenas con comillas simples se almacenan y leen literalmente a como fueron escritas. El siguiente código

```
$literal = 'La $variable no se verá';

print ($literal);
```

producirá el siguiente resultado.

```
Mi $variable no se verá.
```

Las comillas dobles dentro de una cadena limitada por comillas simples no "corta" la cadena, sino que son almacenadas como un carácter norma. La siguiente línea es legal:

```
$literal = 'No estamos usando " para limitar esta cadena';
```

Si se quiere almacenar el carácter ' dentro de la cadena, no es tan complicado, solo hay que *escaparlo*.

```
$literal = 'Esta cadena está limitada por \' ';
```

Comillas dobles

Las cadenas limitadas por comillas dobles son procesadas de dos maneras por el PHP:

1. Algunos caracteres que comienzan con barra invertida ('\') son reemplazados por caracteres especiales.
2. Los nombres de variables (que comienzan con $) son reemplazados por una **representación de cadena** del valor que almacenan.

Los caracteres especiales son:

- \n se reemplaza por un carácter de salto de línea.
- \r se reemplaza por un "retorno de carro" (<ENTER>)
- \t se reemplaza por una tabulación
- \$ se reemplaza por un símbolo dólar ($)
- \" se reemplaza por comillas dobles
- \\ se reemplaza por una barra invertida ('\').

Interpolación de variables

Siempre que aparezca un símbolo $ que no haya sido *escapado*, el PHP intenta interpretarlo como un nombre de variable e inserta el valor de la variable en la cadena. Las reglas son:

- Si la variable tiene como valor una cadena, dicha cadena es insertada en la cadena actual (si está limitada por comillas dobles).
- Si la variable tiene un valor que no sea del tipo `String`, primero se convierte a un `String` y luego es insertado.
- Si la variable no tiene ningún valor asignado, no se inserta nada.

Probar la salida de las siguientes instrucciones:

```
$esto = "esto";

$aquello = "aquello";

$lo_otro = 1.00016

print("$esto, $sin_valor, $aquello+$lo_otro");
```

Arrays

El PHP ofrece la posibilidad de agrupar un conjunto de valores para almacenarlos juntos y referenciarlos por un índice.

Probar la salida del siguiente código:

```
<?
print "Mi_array es $mi_array<BR>";
print "Mi_array[5] es $mi_array[5]<BR>";
$mi_array[5] = "Posición 6ta";
print "Mi_array[5] es $mi_array[5]<BR>";
print "Mi_array es $mi_array<BR>";
?>
```

que produce la siguiente salida:

```
Mi_array es
Mi_array[5] es
Mi_array[5] es Posición 6ta
Mi_array es Array
```

Strings como índices

Los índices puedes ser del tipo numérico (entero) o una cadena de forma indistinta.

```
$comida["Mallorca"] = "Sopas";

$comida["Valencia"] = "Paella";

$comida["Madrid"] = "Cocido";
```

Verificación de Tipos.

gettype(arg)

Retorna un `string` representando el tipo de argumento: `integer`, `double`, `string`, `array`, `object` o `unknown type`.

is_int(arg), is_integer(arg), is_long(arg)

Retorna verdadero si `arg` es de tipo entero, falso en caso contrario.

is_double(arg), is_float(arg), is_real(arg)

Retorna verdadero si `arg` es un `double`, falso en caso contrario.

is_bool(arg)

Retorna verdadero si `arg` es del tipo `Boolean` (`TRUE` o `FALSE`) y falso si no lo es.

is_string(arg)

Retorna verdadero si `arg` es un `string`.

is_array(arg)

Retorna verdadero si `arg` es un `array`.

is_object(arg)

Retorna verdadero si `arg` es un objeto.

Control de Flujo

Es casi imposible hacer programas útiles si no pudiésemos hacer que la ejecución del programa dependiese de determinados valores. Este tipo de ejecución requiere de ***estructuras de control*** que indican que partes del código deben ejecutarse en distintas situaciones.

Hay dos tipos básicos de estructuras de control:

1. Ramificaciones (*branches*): **`if-else-elseif, switch-case`**
2. Ciclos (*loops*): **`while, do-while, for`**

Expresiones Booleanas

Todas las estructuras de control tienen dos partes:

- ***test***, que determina que parte del resto de la estructura se ejecuta. El test funciona evaluando una expresión Booleana.
- ***código***, como una rama separada de código o el cuerpo de un ciclo.

La forma más sencilla de expresiones Booleana son las constantes `TRUE` y `FALSE`:

```
if(TRUE)

echo "Esto siempre se imprime";

else

echo "Esto nunca se imprime";
```

o de forma equivalente:

```
if(FALSE)

echo "Esto nunca se imprime";

else

echo "Esto siempre se imprime";
```

Operadores Lógicos

Los operadores lógicos combinan otras expresiones lógicas para producir un nuevo valor lógico (o Booleano).,

- **`and`**, **`&&`**: Verdadero si ambas condiciones son verdaderas
- **`or`**, **`||`**: Verdadero si una de las condiciones es verdadera.
- **`!`**: Verdadero si la expresión de la derecha es falsa, falso si la expresión de la derecha es verdadera.
- **`xor`**: Verdadero si alguna, pero no ambas, de las condiciones es verdadera.

Operadores de Comparación

- **`==`** (*igual*): Verdadero si ambos argumentos son iguales.
- **`!=`** (*distinto*): Verdadero si ambos argumentos son distintos.
- **`<`** (*menor que*): Verdadero si el argumento de la izquierda es menor que el de la derecha.
- (*mayor que*) Verdadero si el argumento de la izquierda es menor que el de la derecha.
- **`<=`** (*menor o igual*): Verdadero si el argumento de la izquierda es menor o igual que el de la derecha.

- **>=** (*mayor o igual*) Verdadero si el argumento de la izquierda es mayor o igual que el de la derecha.
- **===** (*idéntico*): Verdadero si ambos argumentos son del mismo tipo y tienen el mismo valor.

Ramificación

Las dos estructuras principales son **if** y **switch**. **If** es muy utilizada y es la primera estructura que se aprende. **Switch** es útil para los casos en que tengamos múltiples ramas que dependen de la evaluación de un solo valor.

If-else

```
if (test)

      sentencia-1



if (test)

      sentencia-1

else

      sentencia-2



if($a < $b)

      print ("A es menor que B<br>");

else

      print("B es menor que A<br>");




if($a < $b) {

      print ("A es menor que B<br>");

      print ("A vale $a<br>");

} else

      print("B es menor que A<br>");
```

elseif

El elseif se suele usar cuando tenemos comparaciones en cascada:

```
if($dia == 1)

      print("Primer día");

      else

if ($dia == 2)

                  print("Segundo día");
```

```
            else
```

Se puede reemplazar por:

```
if($dia == 1)
      print("Primer día");
elseif ($dia == 2)
      print("Segundo día");
elseif
```

Haced pruebas con el **if** e intentad poner código HTML en el medio saliendo del modo PHP:

Switch

```
switch ($dia)
{
case: 1:
print("Primer día");
break;
case 2:
print("Segundo día");
break;
}
```

Ciclos

While

```
while(condición)
      sentencia

while(FALSE)
      print("Esta instrucción nunca se ejecuta");

while(TRUE)
      print("Hummm creo que nunca saldremos de este ciclo");
```

ejemplo9.phtml

```
<?
$i = 0;
while($i < 100) {
    print("i vale $i<br>");
    $i = $i + 1; // es equivalente $i++;
}
?>
```

For

```
for(expresión inicial;

test de finalización;

expresión de final de loop)

      sentencia
```

ejemplo10.phtml

```
<?
for ($i=0; $i< 100; $i++) {
    print("i vale $i<br>");
}
?>
```

Do-while

El **do-while** es similar al **while** excepto que la verificación se realiza al final del ciclo, es decir que el ciclo se ejecuta **por lo menos una vez**.

```
do sentencia

      while (expresión);
```

ejemplo11.phtml

```
<?
$i = 0;
do {
    print("i vale $i<br>");
    $i = $i + 1; // es equivalente $i++;
} while ($i < 100);
?>
```

Funciones

Todos los lenguajes de programación modernos proveen capacidades de abstracción de procedimientos que facilita enormemente la programación y el mantenimiento del código. El mecanismo de PHP (al igual que C) es la función. Hay dos clases de funciones:

1. Provistas por el propio lenguaje: `phpinf, strtok, exit`
2. Definidas por el propio programador.

La sintaxis básica para el uso (llamado) de funciones es el nombre de la función seguida, entre paréntesis, de una lista de expresiones separadas por coma:

```
nombre_funcion(expresion_1, expresion_2,  , expresion_N)
```

Cuando el PHP encuentra una llamada a una función, primero evalúa las expresiones especificadas como argumentos y usa sus resultados como valores de entrada a la función. Después de la ejecución de la función, si hay algún valor de retorno, es el resultado de la *expresión de llamada a función*.

Todas las llamadas a funciones son expresiones de PHP, y como cualquier otra expresión, hay dos razones por las que puede interesar llamar una función: por el valor que devuelve o por su efecto secundario.

El valor devuelto por una función es el valor de la expresión, se puede hacer lo mismo que se hace con una expresión normal.

```
$resultado = sqrt(25);

$resultado = sqrt(5 * 5) + sqrt($resultado);
```

Definición de Funciones

Las funciones de usuario no son obligatorias en PHP, sin embargo facilitan la programación en caso que el código se vuelva extenso o requiera la ejecución de tareas complejas. Un función es un trozo de código al cuál se le da un nombre y que puede ser llamada una o varias veces desde distintas partes del programa. La sintaxis es:

```
function nombre_de_funcion ($argumento_1, $argumento_2,  , $argumento_N)

{

    sentencia_1;

    sentencia_2;



}
```

La palabra reservada *function* indica el inicio de la definición de la función. El nombre de la función debe ser construido de forma similar a las variables, puede estar formado de letras, números y "_" y no debe comenzar con un número. Los argumentos son variables de ámbito local en la función. Las operaciones que hace el intérprete cuando encuentra una llamada a función son:

1. El PHP busca la función por el nombre, si no está definida genera un mensaje de error.
2. Sustituye los valores de las expresiones en las llamadas en las variables indicadas como argumentos de la función. Los valores son pasados por copia, no por referencia.
3. Las sentencias en el cuerpo de la función son ejecutadas. Si alguna de las sentencias ejecutadas es un return, devuelve ese valor, caso contrario la función finaliza en la última sentencia ejecutada y no devuelve ningún valor.

Lista de enlaces de este artículo:

1. http://mnm.uib.es/~gallir/php

E-mail del autor: gallir _ARROBA_ uib.es
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=655