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

Manual de PHP 4 en Català (14165 lectures)

Per **Jordi Llonch**, ***lazi*** (http://www.laigu.net)

Creado el 20/11/2001 02:09 modificado el 20/11/2001 02:09

El PHP és un llenguatge per a programació d'aplicacions web. Potser també ho haureu sentit anomenar com a ASP. Encara que aquest nom, en principi, fa referència a tecnologia de l'amic Bill. Pels qui tingueu ganes i temps d'apendre a programar en PHP aquí teniu un manual que a més és en català.

Manual sencer: Manual PHP4 Català[1]

Manual paginat a Bulma: (llegir més)

Manual de PHP 4

Jordi Llonch (jordi@laigu.net[2]**)**

19 de novembre del 2001

Introducció al PHP

PHP, és l'acrònim de "PHP: Hypertext Preprocessor", encara que originariament significava *Personal Home Page Tools.*

És un llenguatge interpretat d'alt nivell empotrable a pàgines HTML.
PHP serveix principalment per a realitzar pàgines html dinàmic de forma senzilla, encara que es pot extendre el seu ús per a shell-scripts, donat que existeixen intèrprets en línea de comana.

Per html dinàmic entendrem al generat o preprocessat per l'intèrpret de PHP que com a sortida ens proporciona una pàgina en html.

PHP és molt semblant a C. Bàsicament es diferencia en que: PHP no és sensible a majúscules i minúscules, excepte en el tractament d'objectes i variables i en que no es declaren les variables i no tenen un tipus fixe, una variable pot emmagatzemar diversos tipus de dades i al llarg de la seva vida aquests poden anar variant.

D'altre banda, cal tenir en compte que PHP s'executa en un servidor i sobre demanda (normalment *submits)* pel que canvia la filosofia de les aplicacions convencionals amb una interfície amb l'usuari molt orientada a events.

Exemple PHP i HTML

El codi PHP serà processat i substitüit per la seva sortida estàndar en el document HTML que s'envia al client (navegador)

El resultat de la sortida estàndar d'aquest còdi serà escrit en la mateixa posició de la pàgina html..

Exemple:

```
<HTML><BODY>
<?php
echo("Hola Gent!<BR>");
?>
</BODY>
</HTML>
```

La sortida d'aques codi seria:

```
<HTML><BODY>
Hola Gent!<BR>
</BODY>
</HTML>
```

Aquest codi guardat com a exemple.phtml o exemple.php i carregat en el navegador, produeix com a resultat una pàgina HTML amb el texte "Hola Gent!".
Podem comprovar que PHP s'executa en el servidor i en el navegador del client només apareix codi HTML, així el client no té cap possibilitat de veure quin codi a produït el resultat en HTML rebut.

Cal dir que per a que funcioni, és necessari tenir instal·lat un servidor web amb suport per a PHP i associar la interpretació de PHP a l'extensió phtml o php. (aquí podem trobar un exemple de configuració d'un servidor Apache en un sistema Unix, http://www.php.net/manual/es/install-unix.php[4])

Que es pot fer amb PHP?

A nivell més bàsic, PHP pot fer qualsevol cosa que es pugui fer amb un script CGI, com processar la informació de formularis, generar, pàgines amb contingut dinàmic, enviar i rebre cookies o interectuar amb el sistema: borra arxius, crea...

A més la majoria de les funcions més útils ja estan predefinies:

- Connectivitat: HTTP, FTP, COM, YP/NIS, SNPM, Sockets, CORBA, LDAP.
- Serveis de correu i notícies: POP, IMAP, SMTP, NNTP.
- Texte i gràfics: XML, HTML, PDF, GD, Flash.
- Funcions matemàtiques.
- POSIX: semàfors, memòria compartida, accés a fitxers, expressions regulars, cronómetres.
- Comerç electrònic: Cybercash, Verisign.
- Formularis.
- Encriptació i compressió: MD5, Gzip, Bzip2, OpenSSL.

Però unes de les característiques més destacades és el suport per a un gran nombre de bases de dades. Escriure una interfícies via web per a una base de dades és tasca senzilla amb PHP. Algunes d'aquestes bases de dades suportades són:

- dBase
- Adabas D
- IBM DB2
- Informix
- InterBase
- mSQL
- MySQL
- ODBC
- Oracle (OCI7 i OCI8)
- PostgreSQL
- SyBase
- ...

PHP també suporta l'ús d'altres serveis que utilitzin protocols com IMAP, SNMP, NNTP, POP3, HTTP, IRC i derivats. També es poden obrir sockets de xarxa directes (raw sockets) i interectuar amb altres protocols.

Referència del llenguatge

PHP i HTML

Per escriure codi en PHP tenim diverses alternatives:

```
1. <? echo ("aquesta és la més simple, una instrucció de processat SGML\n"); ?>
2. <?php echo("si es vol fer servir documents o sintaxi XML, fes-ho així\n"); ?>
3. <script language="php">
     echo ("alguns editors (com FrontPage) no els hi agrada les instruccions de preprocessat");
   </script>
4. <% echo ("També es poden utilitzar etiquetes tipu ASP"); %>
5. <%= $variable; # Forma abreviada de "<%echo .." %>
```

La primera forma només està disponible si s'han habilitat les etiquetes curtes. Es pot fer a través de la funció *short_tags()*, habilitant la opció de configuració *short_open_tag* a l'arxiu de configuració de PHP (*/etc/php4/apache/php.ini*), o bé compilant PHP amb la opció *--enable-short-tags* en el *configure*. (http://www.php.net/manual/es/configuration.php[5])
La quarta forma està disponible si s'han habilitat les etiquetes tipus ASP amb la opció de configuració *asp_tags*.

Separació d'instruccions[6]

Les instruccions es separen igual que en C o Perl, acabant cada sentència amb un punt i coma (;).

L'etiqueta de tancament (?>) també implica el final de la sentència.

Comentaris[7]

Els comentaris en PHP s'escriuen:

- `Amb // o # per a comentaris d'una línia.`
- `Entre /* i */ per a comentaris d'una o més línies.`

Exemple:

```
<?php
// Sortida de texte
/* Aplicació: Aplicació exemple en PHP
Data: Avui
*/
// Sortida de texte
echo("Hola Gent!<BR>");
?>
```

L'exemple només mostra:

```
Hola Gent!
```

Variables[8]

Totes les variables comencen amb el caràcter dólar "$" seguit del nom de la variable no puguent contenir cap punt (.) ni caràcter protegit (?, ¿, $, !, ¡...).

Els noms de les variables són sensibles a majúscules i minúscules

Declaració

No cal declarar les variables, simplement els hi donem un valor per a utilitzar-les:

```
<?php
$strCadena = "Hola Gent";
echo($strCadena);
?>
```

En principi podem intentar accedir a una variable no inicialitzada, PHP interpretarà que no té cap valor és a dir que està buida. També podem configurar el PHP perquè ens emeti un avís en cas que no estigui inicialitzada.

Tipus[9]

Els tipus bàsics de PHP són: Integer, Double, String, Array i Object. Les variables booleanes no existeixen com a tals, sinó que qualsevol valor numèric diferent de 0 o qualsevol cadena no buida es considera *TRUE*. També podem utilitzar les constants *TRUE* o *FALSE* per indicar el valor d'una variable booleana.

Les variables no tenen un tipus fixe, depenen de la última assignació realitzada, adoptant un o altre tipus. La funció *gettype(nom_var)*[10] permet obtenir el tipus d'aquesta variable en forma de cadena:

```
<?php
$variable = "Una cadena";
echo(gettype($variable));
$variable = 0;
echo(gettype($variable));
?>
```

L'exemple escriu "String" i "Integer" per pantalla.

També disposem d'altres funcions per a obtenir el tipus d'una variable: *Is_Double($varname)*[11], *Is_Array($varname)*[12], *Is_String($varname)*[13] y *Is_Object($varname)*[14].

Enters[15]

Els *enters* es poden especificar utilitzant la següent sintaxi:

```
$a = 1234; # número decimal
```

```
$a = -123; # número negatiu
$a = 0123; # número octal (equivalent al 83 decimal)
$a = 0x12; # número hexadecimal (equivalent al 18 decimal)
```

Coma flotant[16]

Els números en coma flotant ("*double*") es poden especificar utilitzant:

```
$a = 1.234; $a = 1.2e3;
```

Cadenes[17]

Les cadenes en PHP s'especifiquen entre cometes simples o cometes dobles:

```
<?php
$strCadena1 = "Hola Gent<BR>";
echo($strCadena1);
$strCadena2 = 'Hola Gentada<BR>';
echo($strCadena2);
?>
```

Existeix una diferència entre un i altre:

```
<?php
$strMissatge = "Hola Gent";
$strMsg = "$strMissatge<BR>";
echo($strMsg);
$strMsg = '$strMissatge<BR>';
echo($strMsg);
?>
```

Es produeix una pàgina amb el texte:

```
Hola Gent
$strMissatge
```

És a dir, quant emprem cometes dobles, les expressions de l'estil $nom_var es substitueixen pel valor de la variable $nom_var, mentre que quant utilitzem cometes simples, la cadena no s'evalua i es deixa com està.

L'operador per a unit cadenes és el punt ".":

```
<?php
$strCadena = "Hola";
$strCadena = $strCadena . " Gent";
echo($strCadena);
?>
```

Les cometes poden contenir més d'una línea sense cap problema:

```
<?php
$strConsulta = '
SELECT *
FROM
taula
WHERE
nom = \'Jordi\';
';
?>
```

Com es pot comprovar, es permet *escapar* les cometes amb la combinació \'. Aquí tenim una taula de caràcters protegits per a les cadenes que estiguin entre cometes dobles (").

seqüència	significat
\n	nova línea
\r	retorn de carro
\t	tabulació horitzontal
\\	barra invertida
\$	signe dòlar
\"	cometes dobles
\[0-7]{1,3}	la seqüència de caràcters que coincideixi amb l'expressió regular és un caràcter en notació octal
\x[0-9A-Fa-f]{1,2}	la seqüència de caràcters que coincideixi amb l'exprèssió regular és un caràcter en notació hexadecimal

Un altre forma de delimitar cadenes és utilitzant la sintaxi de document incrustat (">>>"). S'ha de proporcionar un identificador després de >>>, per tal de que l'intèrpret PHP sàpiga que el texte a incrustar s'ha acabat.
Aquí un exemple:

```
<?php
$str = >>>EOD
Exemple de cadena
Expandint múltiples línees
utilitzant sintaxi de document incrustat.
EOD;
?>
```

Arrays[18]

Els arrays en PHP ens permeten una gran flexibilitat tant en forma de llistes indexades (*vectors*) com en forma de diccionaris (*hash*).

```
<?php
$arrValors[0] = 1;
$arrValors[1] = "Una cadena";
echo("Primer valor: $arrValors[0]<BR>");
echo("Segon valor: $arrValors[1]<BR>");
?>
```

Si no es posa el subíndex de l'element, el valor s'assigna a la següent posició lliure de l'array. Tindrem en compte que els arrays comencen a la posició 0.

```
<?php
$arrValors[] = 1;
$arrValors[] = "Una cadena";
echo("Primer valor: $arrValors[0]<BR>");
echo("Segon valor: $arrValors[1]<BR>");
?>
```

Una altra alternativa per a crear arrays és mitjançant la instrucció d'Array():

```
<?php
$arrValors = Array(1, "Una cadena");
echo("Primer valor: $arrValores[0]<BR>");
echo("Segon valor: $arrValores[1]<BR>");
```

```
?>
```

Fins ara hem tractat els arrays com una llista, però existeix una forma força interessant de direccionar elements que consisteix en l'ús de diccionaris o el que és el mateix, fer un direccionament associatiu mitjançant una clau o *key* (el cas de les claus sí són sensibles a majúscules i minúscules).

```
<?php
$arrValors["nom"] = "Pere";
$arrValors["Cognoms"] = array("Garcia", "Gil");
echo("Nom: $arrValors["nom"]<BR>");
echo("Cognoms: {$arrValors["Cognoms"][0]} {$arrValors["Cognoms"][1]}<BR>");
?>
```

A l'exemple es demostra que es pot fer ús de les dues tècniques de construcció d'arrays, podent tenir un array multidimensional mixte entre llistes i diccionaris.
Adonem-nos que l'array dels cognoms el posem entre claus "{}". És així per evitar ambigüetats, doncs és diferent aquest cas: *{$arrValors["Cognoms"][0]}* d'aquest altre: *{$arrValors["Cognoms"]}[0]*. El primer fa referència al primer element de l'array i en el segon, el *[0]* l'interpretaria com a part de la cadena.

La construcció *array()* també pot utilitzar-se amb arrays associatius:

```
<?php
$arrValors=array(
  "nom" =>"Pere",
  "Cognoms" =>array("Garcia", "Gil")
);
?>
```

La construcció *list()* permet assignar els valors d'un array a una sèrie de variables de cop:

```
<?php
$arrValors=Array(1, "Una cadena", 1.2);
list($nNumber, $strCadena, $fNumber) = $arrValors;
echo("\$nNumber val $nNumber, \$strCadena val " .
"'$strCadena' i \$fNumber val $fNumber");
?>
```

el resultat és:

```
$nNumber val 1, $strCadena val 'Una cadena' i $fNumber val 1.2
```

Conversions[19]

Per a convertir una variable d'un tipus a un altre s'utilitza el *cast* mitjançant parèntesi:

```
<?php
$strVariable = "5";
$valor = (integer) $strVariable;
?>
```

$valor contindrà el valor numèric de la variable $strVariable.

També es pot emprar la funció *SetType($nom_var, "vartype")* per a forçar que la variable *$nom_var* sigui del tipus *vartype*.

De tota manera, PHP és força conseqüent pel que fa als tipus, així que si sumem un número a una cadena, aquesta cadena es converteix en un número.

En una concatenació d'una cadena i un número, la conversió la pateix en número que esdevé cadena de forma automàtica.

Variables predeclarades HTTP[20]

Existeixen unes variables predeclarades que tenen a veure amb el servidor web, són les següents:

- $PHP_AUTH_USER: Usuari de la autenticació.
- $PHP_AUTH_TYPE: Tipus d'autorització.
- $PHP_AUTH_PW: Contrasenya amb la que s'ha fet l'autentificació.
- $HTTP_POST_VARS: Array amb les variables d'un form passades pel mètode POST.
- $HTTP_GET_VARS: Array amb les varibles d'un form passades pel mètode GET.

També podem accedir a les variables provinents d'un formulari directament pel nom de les mateixes, per exemple, tenim aquest form:

```
<form action="envia_form.php" method="post">
    Nom: <input type="text" name="nom"><br>
    <input type="submit">
</form>
```

Al fer el submit, l'script que hi podria haver com a *enviar_form.php* podria tenir aquest aspecte:

```
<?php
echo "En el formulari s'ha escrit el nom $nom<BR>";
?>
```

Es comprova que a la variable *$nom* hi ha el valor del que s'ha introduït al formulari.

Variables variables [21]

Si, el nom és correcte. Les variables variables són formes d'indireccionament a l'hora de referir-se a variables.

Si tenim dues variables *$strVarName* i *$nValue* i fem que *$strVarName* contingui la cadena "*nValue*", al fer referència a *$$strVarName*, ens estarem referint a la variable que té com a nom el contingut de *$strVarName*, és a dir; a *$nValue*.

Exemple:

```
<?php
$nValue = 5;
$strVarName = "nValue";
echo("El valor de $strVarName: $$strVarName.<BR>");
$$strVarName = 5;
echo("Ara $strVarName val $$strVarName.<BR>");
?>
```

En els casos en els que hi hagi ambigüetats, pot emprar-se l'agrupador *{}*, per exemple, per *$$myarray[0]*, podriem referir-nos:

Per indicar el primer element de la variable amb nom el valor que conté *$myArray* fariem: *${$myArray}[0]*

I per referir-nos a la variable amb nom el valor que conté *$myArray[0]* fariem: *${$myArray[0]}* en el segon.

Comprobació de declaració

Per a comprovar si una variable a estat declarada (per exemple el cas de variables que provenen de formularis) podem utilitzar la funció *IsSet($nom_variable)*.

```
<?php
if (isset($but_afegir))
  echo "Anem a afegir!<BR>";
if (isset($but_borrar))
  echo "Anem a borrar!<BR>";
?>
```

Ens pot ser útil alhora de treballar amb formularis, per exemple per saber si el botó que s'ha utilitzat per a fer el submit ha estat el d'afegir o el de borrar.

Constants[22]

Les constants en PHP són literals que no comencen per "$" i que s'inicialitzen amb la construcció: *define(nom_const)*

```
<?php
define("MAX_CLIENTS", 25);
echo(MAX_CLIENTS);
?>
```

Les constants predefinides __FILE__ i __LINE__ ens donen el nom de l'arxiu que conté el codi i el número de línea actual.

Operadors[23]

Operadors Aritmètics[24]

exemple	nom	resultat
$a + $b	Addició	Suma de $a i $b
$a - $b	Substracció	Diferència entre $a i $b
$a * $b	Multiplicació	Producte de $a i $b
$a / $b	Divisió	Quocient de $a entre $b
$a % $b	Mòdul	Reste de $a dividit entre $b

Operadors d'Assignació[25]

L'operador bàsic d'assignació és "=". Aquest operador fa que l'operand de l'esquerra prengui el valor de l'expressió de la dreta.

```
$a = 3;
```

A més, per a totes les operacions aritmètiques i de cadenes que siguin binaries, existeixen els "operadors combinats":

```
$a = 3;
$a += 5; // estableix $a a 8, com si haguéssim escrit: $a = $a + 5;
```

```
$b = "Hola ";
$b .= "Ah"; // estableix $b a "Hola Ah"
```

Operadors a nivell de bits[26]

Permet posar determinats bits d'un enter a 0 ó 1.

exemple	nom	resultat
$a & $b	I	S'activen els bits que estan actius tant en $a com en$b (multiplicació de bits)
$a | $b	O	S'activen els bits que estan actius en $a o en $b (suma de bits)
$a ^ $b	Xor ("O exclusiu")	S'activen els bits que estan actius en $ao en $b però no en els dos alhora
~ $a	No	S'activen els bits que no estan actius en $a
$a << $b	Desplaçament a l'esquerra	Desplaça els bits de $a, $b posicions cap a l'esquerra (per aritmètica binària cada posició d multiplicar per dos el valor d'$a)
$a >> $b	Desplaçament a la dreta	Desplaça els bits de $a, $b posicions cap a la dreta (per aritmètica binària cada posició des dos el valor d'$a)

Operadors de Comparació[27]

Els operadors de comparació, com el seu nom indica, permeten comparar dos valors.

exemple	nom	resultat
$a == $b	Igualtat	Cert si $a és igual a $b
$a === $b	Identitat	Cert si $a és igual a $b i si són del mateix tipus
$a != $b	Desigualtat	Cert si $a no és igual a $b
$a < $b	Menor que	Cert si $a és estrictament menor que $b
$a > $b	Major que	Cert si $a és estrictament major que $b
$a <= $b	Menor o igual que	Cert si $a és menor o igual que $b
$a >= $b	Major o igual que	Cert si $a és major o igual que $b

Un altre operador condicional és l'operador "?:" (o ternari), que funciona com en C i altres llenguatges.

```
(expr1) ? (expr2) : (expr3);
```

L'expressió pren el valor expr2 si expr1 s'evalua com a cert, sinó pren el valor d'expr3.

Operadors d'execució[27]

PHP suporta un operador d'execució: l'apòstrof invertit (`). L'intèrpret intentarà executar la instrucció com si fos una comana de l'intèrpret de comanes (*shell*); i la seva sortida serà retornada com el resultat d'aquesta expressió.

```
<?php
$output = `ls -al`;
echo "<pre>$output</pre>";
?>
```

Operadors d'Increment/decrement[28]

exemple	nom	efecte
++$a	Preincrement	Incrementa $a en un i després retorna $a.
$a++	Postincrement	Retorna $a i després incrementa $a en un.
--$a	Predecrement	Decrementa $a en un i després retorna $a.
$a--	Postdecrement	Retorna $a i després decrementa $a en un.

Operadors Lògics[29]

exemple	nom	efecte
$a and $b	I	Cert si tant $a com $b són certs
$a or $b	O	Cert si $a o $b són certs
$a xor $b	O exclusiva	Cert si $a o $b són certs, però no tots dos alhora
!$a	Negació	Cert si $a no és cert
$a && $b	I	Cert si tant $a com $b són certs
$a || $b	O	Cert si $a o $b són certs

Estructures de control[30]

Les estructures de control són les mateixes que en C, amb algun afegit.

Condicionals[31]

Permet l'execució de codi condicional.

```
if ($nom == "Jordi") {
  // codi si es l'expressió és certa
} elseif ($nom == "Joan") {
  // codi sinó és certa la primera condició però si la segona
} else {
  // codi sinó ha estat certa cap de les dues condicions anteriors
}
```

Switch [32]

Es l'equivalent a una sèrie de sentencies IF.

```
switch ($nom) {
  case "Jordi":
    // Codi per a l'opció Jordi
  break;
  case "Joan":
    // Codi per a l'opció Joan
  break;
  default:
    // Codi per defecte en cas que no s'hagi trobat cap coincidència
}
```

- L'expressió de selecció del *case* ha de ser escalar (no objecte o array).

Iteracions (Repeticions)

En les iteracions es poden utilitzar les instruccions *break* i *continue* per sortir de la iteracció actual o per avançar fins a la propera iteració.

For [33]

```
for ($i=0;$i<10;$i++) {
  echo("Valor actual: $i<BR>");
}
```

Aquest exemple repeteix el codi interior fins a 10 vegades. Veiem que la sortida de la interació és condicionada per l'expressió $i<10.

Foreach[34]

Aquesta construcció és la més senzilla per iterar per un array. Hi han dues sintaxis posibles:

```
foreach(array_expressio as $valor) instrucció

foreach(array_expressio as $clau => $valor) instrucció
```

La primera itera per l'array donat per *array_expressio*. En cada iteració l'element actual s'assigna a *$valor* i el punter intern de l'array avança un element.

La segona sintaxi fa el mateix, execpte que la clau de l'element actual serà assignada a la variable *$key* en cada iteració.

While[35]

```
$bDoExit = 0;
while (!$bDoExit) {
```

```
  echo("Iterant<BR>");
  $bDoExit = 1;
}
```

L'exemple repeteix el cicle mentre es segueixi cumplint l'expressió del *while*. En aquest cas només itera una sola vegada.

Do-While[36]

```
do {
 echo("Iterant<BR>");
 $bDoExit = 1;
} while (!$bDoExit);
```

És el mateix exemple que l'anterior però amb una altre estructura. Com a mínim s'executa una vegada. Comprova l'expressió una vegada interpretades totes les instruccions. El primer ho feia de forma contraria.

List[37], Each[38] (arrays)

Són dos funcions que les podem combinar per iterar per tots els elements d'un array.

```
$arrCognoms = array("Pere" =>"Garcia", "Pol" =>"Gil");
while ( list($strNom, $strCognoms) = each($arrCognoms)) {
  echo("El cognom d'en $strNom és $strCognoms<BR>");
}
```

La sortida seria:

```
El cognom d'en Pere és Garcia
El cognom d'en Pol és Gil
```

Tots els arrays mantenen un comptador intern (accessible mitjançant les funcions *current, reset, next* i *prev*. La funció *each* retorna el *current* i crida a *next*. Amb la funció *list* assignem la clau i l'element a les variables *$strNom* i *$strCognoms*, fins que no queda cap element (*each* llavors retorna *null*).

Array_Walk[39]

Array_Walk és una funció que pren com a paràmetres un array i una funció i aplica aquesta funció a cada element de l'array:

```
<?php
function mostra($elem) {
 echo("$elem<BR>");
}
$arr = array("blau", "negre", "groc", "verd");
Array_Walk($arr,"mostra");
?>
```

La sortida seria:

```
blau
negre
groc
verd
```

Funcions (40)

Declaració

Exemple:

```
<?php
function outputcol($strCadena, $strColor) {
  // Mostra una cadena del color establert
  echo("<FONT COLOR=\"#$strColor\">$strCadena</FONT>");
}
?>
```

La crida és fa com segueix:

```
<?php
outputcol("Vermell", "FF0000");
outputcol("Verd", "00FF00");
?>
```

Paràmetres(41)

Paràmetres per defecte

Si desitgem que la funció per defecte posi el texte en color blau, la redefinirem com segueix:

```
<?php
function outputcol($strCadena, $strColor="0000FF") {
// Mostra una cadena del color establert
  echo("<FONT COLOR=\"#$strColor\">$strCadena</FONT>");
}
?>
```

i la crida llavors podria ser:

```
<?php
outputcol("Defecte");
outputcol("Verd", "00FF00");
?>
```

Els paràmetres predefinits sempre hauran de ser els n últims declarats. No podem posar el valor per defecte al principi dels paràmetres perquè no té sentit fer una crida d'aquest tipus: *outputcol(,"00FF00")*. El PHP ens detectaria un error en aquesta sintaxi.

Paràmetres per referència

Els paràmetres es passen per *valor*, pel que si modifiquem els valors dins de la funció al retorn de la funció els valors no quedaran modificats.

Si desitgem que les variables puguin ser modificades per la funció, farem un pas de paràmetre per *referència*:

```
<?php
function Concatena(&$strDest, $strSrc) {
  // Aquesta funció concatena dos cadenes i les retorna
  // en la primera cadena passada
  $strDest = $strDest . $strSrc;
  // Com que $strSrc no es pasa per referència, la següent
  // instrucció no afecta al paràmetre actual
  $strSrc = "";
}
$strOrigen = " Gent";
$strDesti = "Hola ";
Concatena($strDestino, $strOrigen);
```

```
echo("Origen és $strOrigen i destí és $strDesti<BR>");
?>
```

Es mostra:

```
Origen és Gent y destí és Hola Gent
```

Per a passar un paràmetre per referència, només cal posar "&" davant del nom del paràmetre en la declaració de la funció.

També es pot passar un paràmetre per referència encara que en la funció no estigui declarada com a tal, posant l'*ampersand* "&" al paràmetre actual (al invocar la funció).

Variables en funcions

Variables locals

Per a definir una variable local, simplement s'assigna un valor a la variable:

```
<?php
function calcula($fValor) {
// $fCalc és una variable local
$fCalc = $fValor * 3.14;
}
echo calcula(13.6)."<BR>";
?>
```

Variables estàtiques

Si volem que la variable local conservi el valor d'invocació a cada crida de la funció, la declararem com a estàtica:

```
<?php
function compta() {
  static $compta = 0;
  echo "Comptador: $compta<BR>");
  $compta++;
}
compta();
compta();
compta();
?>
```

Es mostra a la pàgina:

```
Comptador: 0
Comptador: 1
Comptador: 2
```

La inicialització de la variable només es realitza la primera vegada que compta() és invocat.

Acces a variables globals

Per a accedir a una variable global des de dins d'una funció és imprescindible declarar-la dins de la funció com *global $variable*, donat que d'altre forma PHP la prendrà com a variable local.

```
<?php
function tocaGlobal() {
  global $strCadena;
```

```
  $strCadena = "Tocat!";
  $nValue = 7;
  echo("Dins de TouchGlobal \$strCadena val $strCadena i \$nValue val $nValue<BR>");
}
$strCadena = "Hola Gent";
$nValue = 4;
echo("\$strCadena val $strCadena i \$nValue val $nValue<BR>");
tocaGlobal();
echo("\$strCadena ara val $strCadena i \$nValue segueix valent $nValue<BR>");
?>
```

Com es veu, no és necessari que la variable global estigui davant de la funció, tan sols cal que hagi estat inicialitzada abans de cridar a la funció. Els canvis realitzats a una variable global dins d'una funció, romanen quan es retorna de la funció.

Un altre forma d'accedir a les variables globals és mitjançant una indexació associativa de l'array $GLOBALS:

```
$a = 1;
$b = 2;

Function Suma () {
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Suma ();
echo $b;
```

Devolució d'un valor[42]

Per a retornar un valor s'utilitza la clàusula *return*.

```
<?php
function Factorial($nValue) {
  if ($nValue <= 1) {
    return 1;
  } else {
    return Factorial($nValue-1)*$nValue;
  }
}
$nNumber = 6;
echo("El factorial de $nNumber és ".Factorial($nNumber));
?>
```

En PHP les funcions poden cridar-se a si mateixes (*recursivament*), inclús es poden declarar funcions dins de funcions o classes dins de funcions.

Llibreries

Include[43]

Inclou i evalua l'arxiu especificat.

També s'utilitza per incloure HTML.

```
include ('arxiu.inc');
```

Require[44]

Es reemplaça la instrucció *require("nom_arxiu")* per l'arxiu. La diferència amb el primer és que en aquest no s'evalua el contingut de l'arxiu requerit. És el que es sol utilitzar per incloure llibreries externes i és el més ràpid.

```
 require ('lib_form.php');
```

- Tambe existeix en unes construccions complementaries que aseguren que un arxiu només s'inclourà una sola vegada, són *include_once*[45] i *require_once*[46]. Aquestes construccions són molt útils alhora d'utilitzar una llibreria, doncs volem evitar l'error d'incloure una mateixa llibreria dues vegades, ja que provoca un error al redefinir una funcio amb el mateix nom dues vegades.
- Un altre punt que s'ha de tenir molt present és el fet que al incloure un fitxer es surt del mode PHP i per tant si el fitxer inclós conté codi PHP, el mateix haurà de tenir les etiquetes PHP (<?php i ?>).

Classes (Objectes)[47]

Amb PHP podem desenvolupar una programació orientada a objectes. Per aquest motiu tenim les classes.

Una classe és una colecció de variables i de funcions.

Per declarar una classe en PHP utilitzem la construcció *class.*

Les variables de la classe (variables d'instància) es declaren posant *var* abans del nom.

Les funcions membre o mètodes es defineixen dins del bloc de la classe com funcions normals.

Un constructor es defineix amb el mateix nom que l'objecte.

Exemple:

```
<?php
class Carta {
  var $items = 0;  // varible d'instància

  function Carta() { // constructor de la classe
    $this->afegir_item('0', 'primer_element');
  }

  function afegir_item($article, $strNom="sense_nom") { // mètode
    $this->items[$article] = $strNom;
  }

  function borrar_item($article) {  // mètode
    unset($this->items[$article]);
  }
}
?>
```

Com veiem, en les funcions membre podem emprar paràmetres per defecte i qualsevol cosa que s'utilitzaria en una funció normal.

Per utilitzar aquesta classe, fariem:

```
<?php
$carta1 = new Carta();
$carta1->afegir_item('pilota0001', 'Pilota de Basquet');
?>
```

Aquest codi declara un objecte amb el nom *carta1* i hi afegeix un item.

Hem de saber que al fer la declaracio de l'objecte, ja es fa la crida al constructor (funcio *Carta*, dins de la classe), aquesta funcio afegeix un primer item nomes crear l'objecte.

Molt de compte amb les majúscules i minúscules en els noms de les variables de tipus objecte, perquè el PHP en aquest cas, és sensible a majúscules i minúscules.

Les classes poden ser extensions d'altres classes. Les classes exteses o derivades tenen totes les variables i funcions de la classe base i les que s'hi afegeixin al extendre la definició de la classe heredada. La herencia multiple no esta suportada.

```
<?php
class Carta_Nom extends Carta {
  var $propietari

  function estableix_propietari($nom) {
    $this->propietari = $nom;
  }

  function afegir_item($article) { // sobrecàrrega d'un mètode
    $article = $this->propietari.'_'.$article;  // s'utilitza una variable filla
    parent::afegir_item($article);  // crida al mètode agegir_item del pare
  }
}
?>
```

L'exemple crea una nova classe amb una nova variable i funcio. També es sobrecarrega la funció *afegir_item* per tal que tots els articles que entrem se'ls hi afegeixi el nom del propietari, en l'exemple es veu com es modifica la variable que conté l'article i després es fa la crida a la funció del pare d'aquesta classe.

- Hem de tenir en compte, que el constructor de la classe pare no es crida quan es crida al constructor de la classe derivada, per a fer-ho hauriem de fer el seguent:

```
<?php
class Carta_Nom extends Carta {
  var $propietari

  function Carta_Nom() {
    $this->Carta();  // crida al constructor pare
  }

  function estableix_propietari($nom) {
    $this->propietari = $nom;
  }

  function afegir_item($article) {
    $article = $this->propietari.'_'.$article;
    parent::afegir_item($article);
  }
}
?>
```

Característiques interessants

Avaluació de variables amb *Eval*[48]

La funció *Eval($strExpr)* permet avaluar l'expressió *$strExpr*, de forma que si conté códi PHP vàlid, aquest serà interpretat. Això permet coses molt flexibles com per exemple *callbacks*:

```
<?php
function mycallback($strParam) {
  echo("Dins del callback<BR>amb paràmetre $strParam");
}

function myfunc($fnCallback) {
  echo("<TABLE><TR><TD>Callback1:</TD><TD>"); // Creació d'una taula
  Eval($fnCallback); // Crida al callback
  echo("</TD></TR><TABLE>"); // Tanquem la taula
}
$strCode = 'global $strParam; mycallback($strParam);';
$strParam = "Sóc el paràmetre del callback";
```

```
myfunc($strCode);
?>
```

Podem emprar una altre tècnica més senzilla per a fer *callbacks*:

```
<?php
function mycallback() {
  echo("Dins del callback.<BR>");
}
$strCallback = "mycallback";
// Cridem al callback
$strCallback();
?>
```

Control d'errors *Error_Reporting* (49)

Mitjançant la funció *Error_Reporting(màscara)* es poden limitar els errors que captura l'intèrpret de PHP i davant els que aborta l'execució del programa d'entre els següents:

- E_ERROR (1)
- E_WARNING (2)
- E_PARSE (4)
- E_NOTICE (8)
- E_CORE_ERROR (16)
- E_CORE_WARNING (32)

Per exemple, amb:

```
Error_Reporting(E_NOTICE | E_WARNING);
```

Es farà que l'intèrpret de PHP no capturi els errors diferents de *NOTICEs* o de *WARNINGs,* perquè puguem tractar-los nosaltres.

És habitual deshabilitar *E_NOTICE* quant s'utilitza la funció *IsSet* per comprovar si s'ha inicialitzat una variable, donat que sinó tindrem un error i l'intèrpret abortarà la execució. (això en cas que tinguem el PHP configurat perquè ens obligui a inicialitzar les variables):

```
<?php
// Deshabilitar notices
$nOldMask = Error_Reporting(~E_NOTICE);
if (IsSet($btnNou)) {
  // S'ha pulsat el botó Nou en el form
  echo("Donant d'alta l'element solicitat.<BR>");
} elseif (IsSet($btnBaja)) {
  // S'ha pulsat el botó de Baixa en el form
  echo("Donant de baixa l'element solicitat.<BR>");
}
// Restaurem la màscara d'error antiga
Error_Reporting($nOldMask);
?>
```

Opcionalment es pot deshabilitar la detecció d'errors per a una sola sentència avantposant l'arroba "@" a la sentència.

Die[50], **exit**[51]

S'utilitzen per acabar l'execució de l'script de forma sobtada.

Die($msg) mostra el missatge *$msg* abans de sortir.

Exit() surt de l'execució de l'script.

Links

Links als documents sobre el que s'ha basat aquest:

http://www.php.net[52]

http://bulma.net/body.phtml?nIdNoticia=215[53]

http://bulma.net/body.phtml?nIdNoticia=655[54]

Lista de enlaces de este artículo:

1. http://www.laigu.net/ManualPHP.html
2. mailto:jordi@laigu.net
3. http://www.gnu.org/copyleft/fdl.html
4. http://www.php.net/manual/es/install-unix.php
5. http://www.php.net/manual/es/configuration.php
6. http://www.php.net/manual/es/language.basic-syntax.instruction-separation.php
7. http://www.php.net/manual/es/language.basic-syntax.comments.php
8. http://www.php.net/manual/es/language.variables.php
9. http://www.php.net/manual/es/language.types.php
10. http://www.php.net/manual/es/function.gettype.php
11. http://www.php.net/manual/es/function.is-double.php
12. http://www.php.net/manual/es/function.is-array.php
13. http://www.php.net/manual/es/function.is-string.php
14. http://www.php.net/manual/es/function.is-object.php
15. http://www.php.net/manual/es/language.types.php#language.types.integer
16. http://www.php.net/manual/es/language.types.float.php
17. http://www.php.net/manual/es/language.types.string.php
18. http://www.php.net/manual/es/language.types.array.php
19. http://www.php.net/manual/es/language.types.type-juggling.php
20. http://www.php.net/manual/es/language.variables.predefined.php
21. http://www.php.net/manual/es/language.variables.variable.php
22. http://www.php.net/manual/es/language.constants.php
23. http://www.php.net/manual/es/language.operators.php
24. http://www.php.net/manual/es/language.operators.php#language.operators.arithmeti
25. http://www.php.net/manual/es/language.operators.assignment.php
26. http://www.php.net/manual/es/language.operators.bitwise.php
27. http://www.php.net/manual/es/language.operators.comparison.php
28. http://www.php.net/manual/es/language.operators.increment.php
29. http://www.php.net/manual/es/language.operators.logical.php
30. http://www.php.net/manual/es/control-structures.php
31. http://www.php.net/manual/es/control-structures.php#control-structures.if
32. http://www.php.net/manual/es/control-structures.switch.php

33. http://www.php.net/manual/es/control-structures.for.php
34. http://www.php.net/manual/es/control-structures.foreach.php
35. http://www.php.net/manual/es/control-structures.while.php
36. http://www.php.net/manual/es/control-structures.do.while.php
37. http://www.php.net/manual/es/function.list.php
38. http://www.php.net/manual/es/function.each.php
39. http://www.php.net/manual/es/function.array-walk.php
40. http://www.php.net/manual/es/functions.php
41. http://www.php.net/manual/es/functions.arguments.php
42. http://www.php.net/manual/es/functions.returning-values.php
43. http://www.php.net/manual/es/function.include.php
44. http://www.php.net/manual/es/function.require.php
45. http://www.php.net/manual/es/function.include-once.php
46. http://www.php.net/manual/es/function.require-once.php
47. http://www.php.net/manual/es/language.oop.php
48. http://www.php.net/manual/es/function.eval.php
49. http://www.php.net/manual/es/features.error-handling.php
50. http://www.php.net/manual/es/function.die.php
51. http://www.php.net/manual/es/function.exit.php
52. http://www.php.net/
53. http://bulma.net/body.phtml?nIdNoticia=215
54. http://bulma.net/body.phtml?nIdNoticia=655

E-mail del autor: jordi _ARROBA_ laigu.net
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=1010