T i t o r i a l
d e
PHP

O titular dos dereitos de autor desta guía é: http://trisquel-blog.com, e publícaos baixo os termos da licenza Creative Commons. Es libre: Para compartilos, para copialos, para distribuílos e para transmitilos, atribuíndo a autoría, de forma especifica, ao autor, sen que suxira que o autor apóiache niso e coa condición de manter sempre esta nota de copyright ©.En caso de alterar, transformar ou ampliar estes traballos, deberás distribuílos só baixo a mesma licenza ou unha semellante.

Que é PHP.

Instalación.

Tipos de datos.

Constantes.

Etiquetas.

Variables.

Arrays e Estructuras de control.

Funcións.

Arquivos.

Programación orientada a obxectos.

Formularios.

Cookies.

Sesións.

Conexión a bases de datos.

Que é PHP:

PHP é o acrónimo de PHP Hypertext Preprocessor, e é, unha linguaxe de programación de uso xeral de script ao lado do servidor orixinalmente deseñado para o desenvolvemento web de contido dinámico. Foi unha das primeiras linguaxes de programación do lado do servidor que se podían incorporar directamente nun documento HTML en lugar de chamar a un arquivo externo que procese os datos. O código é interpretado por un servidor web cun módulo de procesador de PHP que xera a páxina Web resultante. PHP pode ser usado na maioría dos servidores web do mesmo xeito que en case todos os sistemas operativos, bases de datos e plataformas sen ningún custo.
Foi creado orixinalmente por Rasmus Lerdorf en 1995. Actualmente a linguaxe segue sendo desenvolvida con novas funcións polo grupo PHP. Esta linguaxe forma parte do software libre publicado baixo a licenza PHP que é incompatible coa Licenza Pública Xeneral de GNU debido ás restricións do uso do termo PHP.
Un exemplo:

```
<html>
  <head>
   <title> Exemplo básico PHP</title>
  </head>
  <body>
    <?php
      echo 'Hola mundo';
    ?>
 </body>
</html>
```

Para crear unha saída, escribimos o código HTML con certo código PHP embebido (introducido) no mesmo, o cal producirá a saída “Hola mundo”.
O código PHP inclúese entre etiquetas especiais de comezo e final (<?php e ?>) que nos permitirán entrar e saír no modo PHP.
PHP soporta varios tipos de comentarios.
Exemplo:

```
<?php
    echo "Isto é unha proba"; // Isto é o comentario para unha liña
    /* Este é un comentario multiliña
       máis  comentario */
    echo "Unha proba máis"; # Isto é un comentario
?>
```

Instalación:

Neste titorial imos usar só ferramentas libres: Servidor Apache, Base de Datos MySQL e Editor Bluefish. Para editor valería calquera procesador de textos.
Pódese instalar o Servidor Apache seguindo este titorial:http://trisquel-blog.com/?cat=7
Para MySQL serve este enlace:http://trisquel-blog.com/?cat=8
Para PHP5 en Sistemas Linux baseados en Debian:

 # aptitude install php5-common libapache2-mod-php5 php5-cli

Aptitude instalará de xeito automático o módulo PHP5 para Apache 2 xunto con todas as súas dependencias, e logo activarao. Apache ten que ser reiniciado para que os trocos teñan efecto.
Para usar PHP con MySQL hai que instalar a súa extensión:

 # aptitude install php5-mysql

Neste enlace:http://www.php.net/manual/es/install.unix.php hai máis información sobre a

instalación e configuración de PHP.
Para comprobar que temos PHP, no noso editor, escribimos o seguinte código PHP:

```
<?php
  phpinfo();
?>
```

gardamos o ficheiro co nome infophp.php, pode ser calquera nome agás a extensión que obrigatoriamente é .php, imos o navegador e executámolo, a saída é un cadro coa descrición da configuración de PHP.
Para instalar o Editor Bluefish abonda con:

 # aptitude install bluefish

Existe unha aplicación chamada XAMPP que é un servidor independente da plataforma, software libre, que consiste principalmente na base de datos MySQL, o servidor Web Apache e os intérpretes para linguaxes de script: PHP e Perl. O nome provén do acrónimo de X (para calquera dos diferentes sistemas operativos), Apache, MySQL, PHP, Perl. O programa está liberado baixo a licenza GNU e actúa como un servidor Web libre, fácil de usar e capaz de interpretar páxinas dinámicas. Esta dispoñible para GNU/Linux, Solaris, Windows e MacOS X.

Tipos de datos: Integer, Float, Strings, Boolean:

Integer: Valores numéricos enteiros pódense especificar do xeito seguinte:

```
$a =  1234;    # número decimal
$a =  -123;    # un número negativo
$a =  0123;    # número octal
$a =  0x12;    # número hexadecimal
```

Float: Valores numéricos con parte decimal:

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

Strings: Cadeas de caracteres:

```
$str = "Esto es una cadena";   // Asignando unha cadea.
$str = $str . " con máis texto";  // Engadindo á cadea.
$str .= "E un carácter de nova liña ao final.\n"; /*Outro xeito de engadir,
                                                    inclue un carácter de nova
                                                    liña protexido. */
```

Conversión de cadeas: Cando unha cadea se avalía como un valor numérico, o valor resultante e o tipo determínanse como segue.
A cadea avaliarase como un dobre se contén calquera dos carácteres '.', 'e', ou 'E'. En caso contrario, avaliarase como un enteiro.
O valor vén dado pola porción inicial da cadea. Se a cadea comeza con datos de valor numérico, este será o valor usado. En caso contrario, o valor será 0 (cero). Os datos numéricos válidos son un signo opcional, seguido por un ou máis díxitos (que opcionalmente conteñan un punto decimal), seguidos por un expoñente opcional. O expoñente é unha 'e' ou unha 'E' seguidos por un ou máis díxitos. Cando a primeira expresión é unha cadea, o tipo da variable dependerá da segunda expresión.

```
$foo = 1 "10.5";                 // $foo é dobre (11.5)
$foo = 1 "-1.3e3";               // $foo é dobre (-1299)
$foo = 1 "bob-1.3e3";            // $foo é enteiro (1)
$foo = 1 "bob3";                 // $foo é enteiro (1)
$foo = 1 "10 porquiños";          // $foo é enteiro (11)
$foo = 1 "10 porquiños";          // $foo é enteiro (11)
```

```
$foo = "10.0 porcos " 1;          // $foo é enteiro (11)
$foo = "10.0 porcos " 1.0;        // $foo é double (11)
```

Boolean: Serve para asignar valores lóxicos (VERDADEIRO ou FALSO):

```
$certo = TRUE;   //O valor de $certo e verdadeiro
$certo = FALSE;  //Agora e falso
```

Constantes:

PHP define varias constantes e proporciona un mecanismo para definir máis en tempo de execución. As constantes son como as variables, agás que as constantes deben ser definidas usando a función define(), e que non poden ser redefinidas máis tarde con outro valor. As constantes predefinidas (sempre dispoñibles) son:

__FILE__: Nome do arquivo de comandos que está sendo interpretado actualmente. Se se usa dentro dun arquivo que foi incluído ou requirido, entón dáse o nome do arquivo incluído, e non o nome do arquivo pai.

__LINE__: O número de liña dentro do arquivo que está sendo interpretado na actualidade. Se se usa dentro dun arquivo incluído ou requirido, entón dáse a posición dentro do arquivo incluído.

PHP_VERSION: A cadea que representa a versión do analizador de PHP en uso.

PHP_Os: Nome do sistema operativo no cal se executa o analizador PHP.

TRUE: Valor verdadeiro.

FALSE: Valor falso.

E_ERRO: Denota un erro distinto dun erro de interpretación do cal non é posible recuperarse.

E_WARNING: Denota unha condición onde PHP recoñece que hai algo erróneo, pero continuará de todos os xeitos; pode ser capturado polo propio arquivo de comandos.

E_PARSE: O interprete atopou sintaxe inválida no arquivo de comandos. A recuperación non é posible.

E_NOTICE: Ocorreu algo que puido ser ou non un erro. A execución continúa.

Pódense definir constantes adicionais usando a función define().

Exemplo:

```
define("EXEMPLO_DE_CONSTANTE", "Ola mundo.");
echo CONSTANTE; // Amosa Ola mundo.
```

Etiquetas:

O xeito que ten PHP de delimitar onde empeza e onde termina o seu código, entre o do HTML, é por medio das etiquetas de comezo e fin de código.

```
<?php     //Comezo
  .....
?>        //Fin do código
```

Variables:

En PHP, as variables, non fai falta declaralas, represéntanse como un signo de dólar seguido polo nome da variable. O nome da variable é sensible a minúsculas e maiúsculas.

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var"; // produce a saída "Bob, Joe"
```

As seguintes variables son creadas polo propio PHP.

Argv: Array de argumentos pasados ao script. Cando o script se executa dende a liña de comandos, isto dá un acceso, ao estilo de C, aos parámetros pasados en liña de comandos. Cando se lle chama mediante o método GET, conterá a cadea da petición.

Argc: Contén o número de parámetros da liña de comandos pasados ao script (se se executa dende a liña de comandos).
PHP_SELF: O nome do ficheiro que contén o script que se esta executando, relativo ao directorio raíz dos documentos. Se PHP se esta executando como intérprete de liña de comandos, esta variable non está dispoñible.
HTTP_COOKIE_VARS: Un array asociativo de variables pasadas ao script actual mediante cookies HTTP. Só está dispoñible se o seguimento de variables foi activado mediante a directiva de configuración track_vars ou a directiva.<?php_track_vars?>.
HTTP_GET_VARS: Un array asociativo de variables pasadas ao script actual mediante o método HTTP GET. Só está dispoñible se --variable tracking-- foi activado mediante a directiva de configuración track_vars ou a directiva <?php_track_vars?>.
HTTP_POST_VARS: Un array asociativo de variables pasadas ao script actual mediante o método HTTP POST. Só está dispoñible se --variable tracking-- foi activado mediante a directiva de configuración track_vars ou a directiva <?php_track_vars?>.
Ambito das variables:O ámbito dunha variable é o contexto dentro do que a variable está definida, que pode ser global ou local.
As variables globais son aquelas accesibles dende calquera sitio do código PHP mentres que as locais con as que se definen dentro dunha función, logo, só estan dispoñibles dentro dela.
Exemplo:

```
$a = 1; /* ámbito global */
Function Test () {
    echo $a; /* referencia a una variable de ámbito local */
}
Test ();
```

Este script non producirá saída, xa que a orde echo utiliza unha versión local da variable $a, á que non se lle deu ningún valor no seu ámbito.
Con todo:

```
$a = 1;
$b = 2;
Function Sum () {
  global $a, $b;
  $b = $a + $b;
}
Sum ();
echo $b;
```

Producirá 3 de saída. Ao declarar as variables $a e $b como globais dentro da función todas as referencias a tales variables son a versión global.
Variables externas a PHP (GET e POST): Cando se envía un formulario a un script PHP, as variables de dito formulario pasan a estar dispoñibles de xeito automático no script gracias a PHP.
Exemplo:

```
<form action="foo.php3" method="post">
    Name: <input type="text" name="name"><br>
    <input type="submit">
</form>
```

Cando é enviado, PHP creará a variable $name, que conterá o que sexa que se introduciu no campo Name: do formulario.

Arrays e Estructuras de control:

Un array é unha serie de elementos indexados. A diferenza de outras linguaxes, os datos, non teñen porque ser todos do mesmo tipo. Para acceder a cada un dos elementos do array faise por medio do indice. O xeito máis doado de crear un array é como se fora unha variable pero incluíndo o indice entre corchetes, $a[0] = 1; isto creará un array dun só elemento cuxa posición é 0 e que contén o valor 1. O primeiro elemento das matrices en PHP é o 0.

Exemplos:

```
$a[0] = "abc";
$a[1] = "def";
```

Tamén se pode crear simplemente engadindo valores ao array. Cando se asigna un valor a unha variable array usando corchetes baleiros, o valor engadirase ao final do array.

```
$a[ ] = "ola";      // $a[2] == "ola"
$a[ ] = "mundo"; // $a[3] == "mundo"
```

Os arrays pódense ordenar usando as funcións asort(), arsort(), ksort(), rsort(), sort(), uasort(), usort(), e uksort() dependendo do tipo de ordenación que se desexe.

Pódese contar o número de elementos dun array usando a función count().

Pódese percorrer un array usando as funcións next() e prev(). Outra forma habitual de percorrer un array é usando a función each().

Outro xeito de encher un array é:

```
$a = array("Luns","Martes","Mércores");
```

Arrays asociativos: Son aqueles que o índice é unha cadea:

```
$a["color"] = "azul";
$a["sabor"] = "doce";
$a["forma"] = "redondeada";
$a["nome"] = "maza";
```

ou

```
$a=array("color" => "azul", "sabor" => "doce");
```

Arrays multidimensionais:

```
$a = array(
   "maza" =>  array(
        "color"  => "azul",
        "sabor"  => "doce",
        "forma"  => "redondeada"
   ),
   "laranxa" =>  array(
        "color"  => "laranxa",
        "sabor"  => "ácido",
        "forma"  => "redondeada"
    ),
    "plátano" =>  array(
        "color"  => "amarelo",
        "sabor"  => "doce",
        "forma"  => "lineal"
     )
);
echo $a["maza"]["sabor"];    # devolverá "doce"
```

Estruturas de control:

If: Permite a execución condicional de fragmentos de código. Exemplo:

```
if ($a > $b)
        print "a es mayor que b";
```

A sentenza if pódese aniñar de xeito indefinido dentro doutras sentenzas.

Else: A miúdo queremos executar unha sentenza se se cumpre unha certa condición, e unha sentenza distinta se a condición non se cumpre. Exemplo:

```
if ($a > $b) {
        print "a é maior que b";
} else {
        print "a NON é maior que b";
}
```

Elseif: como o seu nome suxire, é unha combinación de if e else. Como else, estende unha sentenza if para executar unha sentenza diferente no caso de que a expresión if orixinal avalíase como FALSE. Non entanto, a diferenza de else, executará esa expresión alternativa soamente se a expresión condicional elseif avalíase como TRUE. Exemplo:

```
if ($a > $b) {
        print "a é maior que b";
} elseif ($a == $b) {
        print "a é igual que b";
} else {
        print "a é maior que b";
}
```

Sintaxe Alternativa de Estruturas de Control: PHP ofrece unha sintaxe altenativa para algunha das súas estruturas de control; a saber, if, while, for, e switch. En cada caso, a forma básica da sintaxe alternativa é cambiar abrir-chave por dous puntos (:) e pechar-chave por endif;, endwhile;, endfor;, ou endswitch;, respectivamente.

```
<? php if ($a==5): ?>
        A é igual a 5
<?php endif; ?>
```

No exemplo de arriba, o bloque HTML "A = 5" aníñase dentro dunha sentenza if escrita na sintaxe alternativa. O bloque HTML mostraríase soamente se $a fóra igual a 5.

A sintaxe alternativa aplícase a else e tamén a elseif. A seguinte é unha estrutura if con elseif e else no formato alternativo:

```
if ($a == 5):
        print "a é igual a 5";
        print "...";
elseif ($a == 6):
        print "a é igual a 6";
        print "!!!";
else:
        print "a non é nin 5 nin 6";
endif;
```

While: A forma básica dunha sentenza while é:

```
while (expr) sentencia
```

O significado dunha sentenza while é simple. Dille a PHP que execute as sentenzas aniñadas, tanto como a expresión while se avalíe como TRUE. O valor da expresión é verificado cada vez ao comezo do bucle, polo que ata se este valor cambia durante a execución das sentenzas aniñadas, a execución non se deterá ata o final da iteración (cada vez que PHP executa as sentenzas contidas no bucle é unha iteración). Se a expresión while se avalíase como FALSE dende o principio, as

sentenzas aniñadas non se executarán nin sequera unha vez.
Como coa sentenza if, pódense agrupar multiples sentenzas dentro do mesmo bucle while encerrando un grupo de sentenzas con chaves, ou usando a sintaxe alternativa:

```
while (expr): sentenza ... endwhile;
```

O seguintes exemplos imprimen números do 1 ao 10:

```
/* Exempol 1 */
$i = 1;
while ($i <= 10) {
        print $i ; /* o valor impreso sería $i antes do incremento (post-incremento) */
        $i++;
}
/* ejemplo 2 */
$i = 1;
while ($i <= 10):
        print $i;
        $i++;
endwhile;
```

Do..while: Os bucles do..while son moi similares aos bucles while, agás que as condicións compróbanse ao final de cada iteración no canto de ao principio. A principal diferenza fronte aos bucles regulares while é que se garante a execución da primeira iteración dun bucle do..while, a condición compróbase só ao final da iteración, mentres que pode non ser necesariamente executada cun bucle while regular, a condición compróbase ao principio de cada iteración, se esta avalíase como FALSE desde o principio a execución do bucle finalizaría inmediatamente).
Hai unha soa sintaxe para os bucles do..while:

```
$i = 0;
do {
    print $i;
} while ($i>0);
```

For: A sintaxe dun bucle for é:

```
for (expr1; expr2; expr3) sentencia
```

A primeira expresión (expr1) avalíase incondicionalmente unha vez ao principio do bucle. Ao comezo de cada iteración, avalíase expr2. Se se avalía como TRUE, o bucle continúa e as sentenzas aniñadas execútanse. Se se avalía como FALSE, a execución do bucle finaliza.
Ao final de cada iteración, avalíase (executa) expr3. Cada unha das expresións pode estar baleira. Que expr2 estea baleira significa que o bucle debería correr indefinidamente. Isto poida que non sexa tan inútil como se podería pensar, posto que a miúdo quérese saír dun bucle usando unha sentenza break condicional no canto de usar a condición de for. Exemplos.
Todos eles mostran números do 1 ao 10:

```
/* exemplo 1 */
for ($i = 1; $i <= 10; $i ) {
        print $i;
}
/* exemplo 2 */
for ($i = 1;;$i ) {
        if ($i > 10) {
        break;
}
print $i;
```

```
}
/* exemplo 3 */
$i = 1;
for (;;) {
        if ($i > 10) {
        break;
}
print $i;
$i ;
}
/* exemplo 4 */
for ($i = 1; $i <= 10; print $i, $i ) ;
```

Por suposto, o primeiro exemplo parece ser o mais elegante (ou quizais o cuarto), pero un pode descubrir que ser capaz de usar expresións baleiras en bucles for resulta útil en moitas ocasións.

Foreach: Ten dúas sintaxes:

```
foreach(expresion_array as $value) sentenza
foreach(expresion_array as $key => $value) sentencia
```

A primeira forma percorre o array dado por expresion_array. En cada iteración, o valor do elemento actual asígnase a $value e o punteiro interno do array avánzase nunha unidade (así no seguinte paso, estarase mirando o elemento seguinte).

O segundo xeito fai o mesmo, salvo que a crave do elemento actual será asignada á variable $key en cada iteración.

As seguintes funcionalidades son idénticas:

```
reset( $arr );
while( list( , $value ) = each( $arr ) ) {
        echo "Valor: $value<br>\n";
}
foreach( $arr as $value ) {
        echo "Valor: $value<br>\n";
}
```

As seguintes tamén son funcionalidades idénticas:

```
reset( $arr );
while( list( $key, $value ) = each( $arr ) ) {
        echo "Key: $key; Valor: $value<br>\n";
}
foreach( $arr as $key => $value ) {
        echo "Key: $key; Valor: $value<br>\n";
}
```

Algúns exemplos máis do seu uso:

```
/* foreach exemplo 1: só valor*/
$a = array(1, 2, 3, 17);
foreach($a as $v) {
        print "Valor actual de \$a:  $v.\n";
}
/* exemplo 2: valor (con clave impresa para ilustrar) */
$a = array(1, 2, 3, 17);
$i = 0; /* só para propósitos demostrativos */
foreach($a as $v) {
```

```
        print "\$a[$i] => $k.\*n";
}
/* exemplo 3: clave e valor */
$a = array(
"un" => 1,
"dous" => 2,
"tres" => 3,
"dezasete" => 17
);
foreach($a as $k => $v) {
        print "\$a[$k] => $v.\n";
}
```

Break: Remata a execución da estrutura actual for, foreach, while, do-while ou switch. Acepta un argumento numérico opcional o cal indica de cantas estruturas aniñadas encerradas se ten que saír. Exemplo:

```
$arr = array('un', 'dous', 'tres', 'catro', 'stop', 'cinco');
while (list(, $val) = each($arr)) {
        if ($val == 'stop') {
                break;  /* Pódese escribir tamén break 1. aqui*/
        }
echo "$val<br> \n";
}
/* Usando o argumento opcional. */
$i = 0;
while ( $i) {
        switch ($i) {
                case 5:
                        echo "En 5<br>\n";
                        break 1; /* Só sae do switch. */
                case 10:
                        echo "En 10; saíndo<br>\n";
                        break 2; /* Sae do switch e do while. */
                default:
                        break;
        }
}
```

Continue: Utilízase dentro das estruturas iterativas para saltar o resto da iteración actual do bucle e continuar a execución na avaliación da condición e logo comezar a seguinte iteración.

En PHP a sentenza switch considérase unha estrutura iterativa para os propósitos de continue. A sentenza continue acepta un argumento numérico opcional, que indica cantos niveis de bucles encerrados se han saltar. O valor por omisión é 1, polo que salta ao final do bucle actual. Exemplo:

```
while (list($clave, $valor) = each($arr)) {
        if (!($clave % 2)) {  // saltar os membros impares
                continue;
        }
        facer_algo($valor);
}
```

```
$i = 0;
while ($i < 5) {
        echo "Exterior<br>\n";
while (1) {
        echo "Medio<br>\n";
while (1) {
        echo "Interior<br>\n";
        continue 3;
}
echo "Isto nunca se imprimirá.<br>\n";
}
echo "Nin isto tampouco.<br>\n";
}
```

Switch: A sentenza switch é similar a unha serie de sentenzas IF na mesma expresión. En moitas ocasións, quérese comparar a mesma variable (ou expresión) con nuchos valores diferentes, e executar unha parte de código distinta dependendo da que valor é igual.
Os dous exemplos seguintes son dous xeitos distintos de escribir a mesma cousa, un usa unha serie de sentenzas if, e o outro usa a sentenza switch:

```
/*exemplo1 */
if ($i == 0) {
        print "i é igual a 0";
}
if ($i == 1) {
        print "i é igual a 1";
}
if ($i == 2) {
        print "i é igual a 2";
}
/*exemplo 2 */
switch ($i) {
        case 0:
                print "i é igual a 0";
                break;
        case 1:
                print "i é igual a 1";
                break;
        case 2:
                print "i é igual a 2";
                break;
}
```

É importante entender como se executa a sentenza switch para evitar erros. A sentenza switch executa liña por liña (realmente, sentenza a sentenza). Ao comezo, non se executa código. Só cando se atopa unha sentenza case cun valor que coincide co valor da expresión switch PHP comeza a executar as sentenzas. PHP continúa executando as sentenzas ata o final do bloque switch, ou a primeira vez que vexa unha sentenza break. Se non se escribe unha sentenza break ao final dunha lista de sentenzas case, PHP seguirá executando as sentenzas do seguinte case.
Include: Esta sentenza inclúe e avalía o arquivo especificado.
O seguinte tamén se aplica a require.

Os arquivos son incluídos con base na ruta de acceso dada ou, se ningunha é dada, o include_path especificado. Se o arquivo non se atopa no include_path, include finalmente verificará no propio directorio do script que fai o chamado e no directorio de traballo actual, antes de fallar. O construtor include emitirá unha advertencia se non pode atopar un arquivo, este é un comportamento diferente ao de require, o cal emitirá un erro fatal.
Se unha ruta é definida xa sexa absoluta (comezando cunha letra de unidade ou \ en Windows ou / en sistemas Unix/Linux) ou relativa ao directorio actual (comezando con . ou ..) o include_path será ignorado por completo. Por exemplo, se un nome de arquivo comeza con ../, o interprete buscará no directorio pai para atopar o arquivo solicitado.
Cando se inclúe un arquivo, o código que contén herda o ámbito das variables da liña na cal ocorre a inclusión. Calquera variable dispoñible nesa liña do arquivo que fai o chamado, estará dispoñible no arquivo chamado, desde ese punto en diante. Con todo, todas as funcións e clases definidas no arquivo incluído teñen o ámbito global.
Exemplo:

```
vars.php  //nome do arquivo a incluír
<?php
        $cor = 'verde';
        $froita = 'mazá';
?>

test.php  // segundo arquivo
<?php
        echo "Unha $froita $cor"; // Unha
        include 'vars.php';
        echo "Unha $froita $cor"; // Unha mazá verde
?>
```

Require: é idéntico a include agás que en caso de fallo producirá un erro fatal de nivel E_COMPILE_ERRO. Noutras palabras, este detén o script mentres que include só emitirá unha advertencia (E_WARNING) o cal permite continuar o script.
Include_once: A sentenza include_once inclúe e avalía o ficheiro especificado durante a execución do script. É un comportamento similar ao da sentenza include, sendo a única diferenza que se o código do ficheiro xa foi incluído, non se volverá a incluír. Como o seu nome indícao, será incluído só unha vez. include_once pode ser usado en casos onde o mesmo ficheiro podería ser incluído e avaliado máis dunha vez durante unha execución particular dun script, así que neste caso, pode axudar a evitar problemas como a redefinición de funcións, reasignación de valores de variables.
Require_once: é idéntica a require agás que PHP verificará se o arquivo xa foi incluído e se é así, non se inclúe (require) de novo.

Funcións:

Unha función pode ser definida usando unha expresión como a seguinte:

```
function foo ($arg_1, $arg_2, ..., $arg_n) {
    echo "Función de ejemplo.\n";
    return $retval;
}
```

A información pode fornecerse ás funcións mediante unhaa lista de parámetros, unha lista de variables e/ou constantes separadas por comas.
PHP soporta pasar parámetros por valor (o comportamento por defecto), por referencia, e parámetros por defecto.

Parámetros por referencia: Por defecto, os parámetros dunha función pásanse por valor, de maneira que se troca o valor do argumento dentro da función, non se ve modificado fóra dela.
Para permitir modificar o valor dos parámetros dunha función, hai que pasalos por referencia, para iso, hai que antepor un ampersand (&) ao nome do parámetro na definición da función.
Exemplo:

```
function add_some_extra(&$string) {
    $string .= ' e algo máis.';      //O punto antes do signo igual concatena as cadeas
}
$str = 'Isto é unha cadea, ';
add_some_extra($str);
echo $str;    // Saca 'Isto é unha cadea, e algo máis.'
```

Para pasar unha variable por referencia a unha función que non toma o parámetro por referencia por defecto, tes que antepor un ampersand ao nome do parámetro na chamada á función:

```
function foo ($bar) {
  $bar .= 'e algo máis.';
}
$str = 'Isto é unha cadea, ';
foo ($str);
echo $str;    // Saca 'Isto é unha cadea, '
foo (&$str);
echo $str;    // Saca 'Isto é unha cadea,  e algo máis.'
```

Parámetros por defecto: Unha función pode definir valores por defecto para os parámetros escalares.
Exemplo:

```
function makecoffee ($type = "cappucino") {
    return "Facer unha cunca de $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
```

A saída do fragmento anterior é:

```
Facer unha cunca de cappucino.
Facer unha cunca de espresso.
```

O valor por defecto ten que ser unha expresión constante, e non unha variable ou membro dunha clase.
Devolver valores: Os valores retórnanse usando a instrución opcional **return**. Pode devolverse calquera tipo de valor, incluíndo listas e obxectos.
Exemplo:

```
function square ($num) {
    return $num * $num;
}
echo square (4); // saca 16.
```

Non se poden devolver múltiples valores desde unha función, pero un efecto similar pódese conseguir devolvendo unha lista.

```
function small_numbers() {
return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
```

Arquivos:

Un arquivo ou ficheiro informático é un conxunto de bits almacenado nun dispositivo. Un arquivo é identificado por un nome e a descrición do cartafol ou directorio que o contén.
Ademais de bases de datos e arquivos HTML, PHP tamén permite o uso de ficheiros de texto. Exemplo:

```
$ruta = "/carpeta/contador.txt";
$fp = fopen($ruta, "w+");
$conta = fread($fp, filesize($ruta));
$conta++;
rewind($fp);
fputs($fp, $conta);
fclose($fp);
```

Para abrir un arquivo úsase a función **fopen()**, cos parámetros **filename** e **mode**, que dará como resultado o punteiro de arquivos (file pointer) que gardaremos na variable $fp.
Filename E o nome do arquivo, no exemplo é contador, que se atopa no cartafol carpeta.
Mode Dinos o que podemos facer cun arquivo:

- r Só para ler, punteiro ao inicio.
- r+ Ler e escribir, punteiro ao inicio.
- w Só para escribir, punteiro ao inicio, se existe borra os datos.
- w+ Ler e escribir, punteiro ao inicio, se existe borra os datos.
- a Escritura, punteiro ao final, se non existe créao.
- a+ Ler e escribir, punteiro ao final, se non existe créao.

A función **fread**(punteiro, tamaño_da_cadea) recupera do arquivo unha cadea de caracteres e asígnallo a variable **$conta** , o segundo argumento é a lonxitude da cadea.
fgets(punteiro) fai o mesmo que fread coa diferenza que interpretará como final de cadea o primeiro retorno de carro que atope. A lonxitude máxima da cadea será de 1024 caracteres.
Unha función semellante e **file_get_contents**($filename) coa diferenza que transmite o arquivo enteiro a unha cadea.
$conta++ súmalle 1 ao valor da variable.
rewind(punteiro) situa o punteiro no principio de arquivo. A función **fseek**(punteiro, 0) fai o mesmo.
fputs(punteiro, variable) encargase de sobre-escribir, argumento w+, o contido de **$conta** no arquivo. Tamén se podería empregar **fwrite**(punteiro, variable). **file_put_contents**($filename, $variable) escribe unha cadea nun arquivo. Esta función é idéntica que chamar a **fopen()**, **fwrite()** e **fclose()** sucesivamente para escribir información nun arquivo.
Por último **fclose**(punteiro) pecha o arquivo.
Isto é un exemplo dun contador de visitas a unha páxina web.

Programación orientada a obxectos:

A programación orientada a obxectos ou **POO** (OOP segundo as súas siglas en inglés) é un paradigma de programación que usa os obxectos nas súas interaccións, para deseñar aplicacións e programas informáticos.
Características:
Abstracción Illamento dun elemento do seu contexto. Define as características esenciais dun obxecto.
Encapsulamento Reúne ao mesmo nivel de abstracción, a todos os elementos que poidan considerarse pertencentes a unha mesma entidade.

Modularidade Característica que permite dividir unha aplicación en varias partes máis pequenas (denominadas módulos), independentes unhas doutras.
Ocultación (illamento) Os obxectos están illados do exterior, protexendo ás súas propiedades para non ser modificadas por aqueles que non teñan dereito a acceder ás mesmas.
Polimorfismo É a capacidade que dá a diferentes obxectos, a posibilidade de contar con métodos, propiedades e atributos de igual nome, sen que os dun obxecto interfiran co doutro.
Herdanza É a relación existente entre dúas ou máis clases, onde unha é a principal (nai) e outras son secundarias e dependen (herdan) delas (clases fillas), onde á vez, os obxectos herdan as características dos obxectos dos cales herdan.
Recolección de lixo É a técnica que consiste en destruír aqueles obxectos cando xa non son necesarios, liberándoos da memoria.
Elementos da POO:
Clase Unha clase é un modelo que se utiliza para crear obxectos que comparten un mesmo comportamento, estado e identidade.
Obxecto É unha entidade provista de métodos ou mensaxes aos cales responde (comportamento), atributos con valores concretos (estado) e propiedades (identidade).

```
$persoa = new Persoa();
```

Método É o algoritmo asociado a un obxecto que indica a capacidade do que este pode facer.

```
function camiñar() {
    .................
}
```

Evento e Mensaxe Un evento é un suceso no sistema mentres que unha mensaxe é a comunicación do suceso dirixida ao obxecto.
Propiedades e atributos As propiedades e atributos, son variables que conteñen datos asociados a un obxecto.

```
$nome = 'Juan';
$idade = '25 anos';
$altura = '1,75 metros';
```

Clases e obxectos en PHP 5:
Segundo o **Manual Oficial de PHP** *unha clase é unha colección de variables e funcións que traballan con estas variables. As variables defínense utilizando* ***var*** *e as funcións utilizando* ***function***.
Exemplo:

```
class Carro {
    var $contido;
    function mete($cousa) {
        $this->contido = $cousa;
    }
    function amosa_contido() {
        return $this->contido;
    }
}
```

O exemplo define unha clase chamada Carro que consiste en ir metendo artigos no carro e dúas funcións (os métodos da clase), **mete** e **amosa_contido**, para meter cousas nel e amosar o contido. A palabra clave **this** precedida do signo do dólar ,“$”, indica que fai referencia ao propio obxecto. Se o método ou a clase teñen que devolver algún valor empregase a palabra reservada **return**.
Declaración de Clases abstractas As clases abstractas son aquelas que non necesitan ser

instanciadas e que serán herdadas nalgún momento. Defínense antepondo a palabra crave **abstract** a **class**:

```
abstract class NomeDaClaseAbstracta {
   .....................................
}
```

Este tipo de clases, será a que conteña métodos abstractos e xeralmente, a súa finalidade, é a de declarar clases "xenéricas" que necesitan ser declaradas pero ás cales, non se pode outorgar unha definición precisa (diso, encargaranse as clases que a herden).

Herdanza de Clases: Os obxectos poden herdar propiedades e métodos doutros obxectos. Para iso, PHP permite a herdanza de clases, cuxa característica representa a relación existente entre diferentes obxectos. Para definir unha clase como extensión dunha clase "nai" utilízase a palabra crave **extends**.

```
class NomeDaClaseMai {
      ......................
}
class NomeDaClaseFilla extends NomeDaClaseMai {
 /* esta clase herda todos os métodos e propiedades da clase nai */
}
```

Declaración de Clases finais: PHP 5 incorpora clases finais que non poden ser herdadas por outra. Defínense antepondo a palabra crave final.

```
final class NomeDaClaseFinal {
  # esta clase non poderá ser herdada
}
```

Obxectos en PHP: Unha vez que as clases foron declaradas, será necesario crear os obxectos, aínda que temos visto que algunhas clases, como as clases abstractas, son só modelos para outras, e polo tanto non necesitan instanciar ao obxecto.

Instanciar unha clase: Para instanciar unha clase, só é necesario utilizar a palabra crave **new**. O obxecto será creado, asignando esta instancia a unha variable (a cal, adoptará a forma de obxecto). Loxicamente, a clase debe ser declarada antes de ser instanciada.

```
# declaración da clase
class Persoa {
    .....
}
# creación do obxecto instanciando a clase
 $persoa = new Persoa();
```

Propiedades: As propiedades representan certas características do obxecto en si mesmo. Defínense antepondo a palabra crave **var** ao nome da variable (propiedade):

```
class Persoa {
   var $nome;
   var $idade;
   var $xero;
}
```

As propiedades poden gozar de diferentes características, por exemplo, a visibilidade: poden ser públicas, privadas ou protexidas. A visibilidade das propiedades, é aplicable tamén á visibilidade dos métodos.

Propiedades públicas: As propiedades públicas defínense antepondo a palabra crave **public** ao nome da variable. Estas, poden ser accedidas desde calquera parte da aplicación, sen restrición.

```
class Persoa {
```

```
    public $nome;
    public $xero;
}
```

Propiedades privadas: As propiedades privadas defínense antepondo a palabra crave **private** ao nome da variable. Estas só poden ser accedidas pola clase que as definiu.

```
class Persoa {
  public $nome;
  public $xero;
  private $idade;
}
```

Propiedades protexidas: As propiedades protexidas poden ser accedidas pola propia clase que a definiu, así como polas clases que a herdan, pero non, desde outras partes da aplicación. Estas, defínense antepondo a palabra crave **protected** ao nome da variable:

```
class Persoa {
   public $nome;
   public $xero;
   private $idade;
   protected $pasaporte;
}
```

Propiedades estáticas: As propiedades estáticas representan unha característica de "variabilidade" dos datos, de gran importancia en PHP 5. Unha propiedade declarada como estática, pode ser accedida sen necesidade de instanciar un obxecto. e o seu valor é estático (é dicir, non pode variar nin ser modificado). Esta, defínese antepondo a palabra crave **static** ao nome da variable:

```
class PersoaAPositivo extends Persoa {
      public static $tipo_sangue = 'A+';
}
```

Acceso ás propiedade dun obxecto: Para acceder ás propiedade dun obxecto, existen varios xeitos de facelo. Todas elas, dependerán do ámbito desde o cal se invoquen así como da súa condición e visibilidade.

Acceso a variables dende o ámbito da clase: Accédese a unha propiedade non estática dentro da clase, utilizando a palabra reservada **$this** sendo esta unha referencia ao obxecto mesmo:

```
$this->nome;
```

Cando a variable é estática, accédese a ela mediante o operador de resolución de ámbito, dous-puntos dobres "::" antepondo a palabra crave **self** ou **parent** segundo se trata dunha variable da mesma clase ou doutra da cal se herdou:

```
print self::$variable_estatica_de_esta_clase;
print parent::$variable_estatica_de_clase_nai;
```

Acceso a variables desde o exterior da clase: Accédese a unha propiedade non estática coa seguinte sintaxe: $obxecto->variable Nótese ademais, que este acceso dependerá da visibilidade da variable. Polo tanto, só variables públicas poden ser accedidas desde calquera ámbito fóra da clase herdada.

```
# creación do obxecto instanciando a clase
$persoa_a_positivo = new PersoaAPositivo();
# acceso á variable NON estática
print $persoa_a_positivo->nome;
```

Para acceder a unha propiedade pública e estática o obxecto non necesita ser instanciado, permitindo así, o acceso a dita variable mediante a seguinte sintáxis: Clase::$variable_estática

```
# acceso á variable estática
```

```
print PersoaAPositivo::$tipo_sangre;
```

Outro tipo de “propiedade” dunha clase, son as **constantes**, aquelas que manteñen o seu valor de forma permanente e sen cambios. A diferenza das propiedades estáticas, as constantes só poden ter unha visibilidade pública.

Pode declararse unha constante de clase como calquera constante normal en PHP 5. O acceso a constantes é exactamente igual que ao doutras propiedades.

```
Const MIÑA_CONSTANTE = 'Isto é o que vale a miña contante';
```

Métodos en PHP: Convén lembrar, para quen veña da programación estruturada, que o método de unha clase, é un algoritmo igual ao dunha función.

A única diferenza entre método e función, é que chamamos método ás funcións dunha clase (na POO), mentres que chamamos funcións, aos algoritmos da programación estruturada.

A forma de declarar un método é antepondo a palabra crave **function** ao nome do método, seguido por un par paréntese de apertura e peche e chaves que pechen o algoritmo:

```
# declaración da clase
class Persoa {
    #propiedades

    #métodos
    function doar_sangue() {
        ..... .
    }
}
```

Do mesmo xeito que calquera outra función en PHP, os métodos recibirán os parámetros necesarios indicando aqueles requiridos, dentro dos paréntisis:

```
# declaración da clase
class Persoa {
    #propiedades

    #métodos
    function doar_sangue($destinatario) {
        .....
    }
}
```

Métodos públicos, privados, protexidos e estáticos: Os métodos, do mesmo xeito que as propiedades, poden ser públicos, privados e protexidos ou estáticos. A forma de declarar a súa visibilidade tanto como as características desta, é exactamente a mesma que para as propiedades.

```
static function a() { }
protected function b() { }
private function c() { }
```

Métodos abstractos: A diferenza das propiedades, os métodos, poden ser abstractos como sucede coas clases.

O Manual Oficial de PHP, refírese aos métodos abstractos, describíndoos do seguinte xeito:

Os métodos definidos como abstractos simplemente declaran a estrutura do método, pero non poden definir a implementación. Cando se herda dunha clase abstracta, todos os métodos definidos como abstract na definición da clase parent deben ser redefinidos na clase child; adicionalmente, estes métodos deben ser definidos coa mesma visibilidade (ou cunha menos restritiva). Por exemplo, se o método abstracto está definido como

protected, a implementación da función pode ser redefinida como protected ou public, pero nunca como private.

Para entender mellor os métodos abstractos, poderiamos dicir que os métodos abstractos son aqueles que se declaran inicialmente nunha clase abstracta, sen especificar o algoritmo que implementarán, é dicir, que só son declarados pero non conteñen un "código" que especifique que farán e como o farán.

Métodos máxicos en PHP 5: PHP 5, tráenos unha chea dos auto-denominados "métodos máxicos". Estes métodos, outorgan unha funcionalidade pre-definida por PHP, que poden achegar valor ás clases e aforrar grandes cantidades de código. Antepoñen ao nome dous guións baixos "__".

Entre os métodos máxicos, podemos atopar os seguintes:

__construct(): O método __construct() é aquel que será invocado de xeito automático, ao instanciar un obxecto. A súa función é a de executar calquera inicialización que o obxecto necesite antes de ser utilizado.

```
# declaración da clase
class Produto {
   #definición de algunhas propiedades
   public $nome;
   public $prezo;
   protected $estado;
   #definición do método set_estado_produto()
   protected function set_estado_produto($estado) {
        $this->estado = $estado;
    }
# construtor da clase
   function __construct() {
       $this->set_estado_produto('en uso');
  }
}
```

No exemplo anterior, o construtor da clase encárgase de definir o estado do produto como en uso, antes de que o obxecto (Produto) comece a utilizarse. Se se agregasen outros métodos, estes, poderán facer referencia ao estado do produto, para determinar se executar ou non determinada función. Por exemplo, non podería mostrarse á venda un produto "en uso polo sistema", xa que a este, poderíaselle estar modificando o prezo.

__destruct(): O método __destruct() é o encargado de liberar da memoria, ao obxecto cando xa non é referenciado. Pódese aproveitar este método, para realizar outras tarefas que se estimen necesarias no momento de destruír un obxecto.

```
# declaración da clase
class Produto {
     #definición de algunhas propiedades
        public $nome;
        public $prezo;
        protected $estado;
        #definición do método set_estado_produto()
        protected function set_estado_produto($estado) {
                $this->estado = $estado;
        }
        # construtor da clase
        function __construct() {
```

```
                $this->set_estado_produto('en uso');
        }
        # destrutor da clase
        function __destruct() {
                $this->set_estado_produto('liberado');
                print 'O obxecto foi destruído';
        }
}
```

Outros métodos máxicos: PHP ofrécenos outros métodos máxicos como __call, __callStatic, __get, __set, __isset, __unset, __sleep, __wakeup, __toString, __invoke, __set_state e __clone.
Pode verse unha descrición e exemplo do seu uso, no sitio web oficial de PHP: http://www.php.net/manual/es/language.oop5.magic.php

Formularios:

Unha das características máis importantes de PHP é que xestiona formularios HTML. O concepto básico que é importante entender é que calquera elemento dos formularios estará dispoñible automaticamente no seu código PHP. Exemplo:

```
/*Ejemplo #1 Un formulario HTML simple */
<form action="accion.php" method="post">
 <p>Seu nome: <input type="text" name="nome" /></p>
<p>Súa idade: <input type="text" name="idade" /></p>
<p><input type="submit" /></p>
</form>
```

Os campos **nome** e **idade** do formulario son os datos de entrada. **Submit** encargase de enviar ao formulario.

```
/*Ejemplo #2 Imprimir información do noso formulario */
Ola <?php echo htmlspecialchars($_POST['nome']); ?>.
Vostede ten <?php echo (int)$_POST['idade']; ?> anos de idade.
```

A función **htmlspecialchars()** asegúrase que todos os caracteres que son especiais en html sexan codificados de maneira que ninguén poida inxectar etiquetas HTML ou Javascript na páxina web. O campo idade, como sabemos que é un número, podemos convertelo nun integer **(int)** que se desfará de calquera carácter non numérico. As variables **$_POST**['nome'] e **$_POST**['idade'] dan acceso aos datos do formulario. Isto é así porque usamos o método **POST**.
Se se usa o método **GET**, os datos enviaranse a través da URL e serán visibles, os datos do formulario estarían en **$_GET**. No seu lugar tamén se pode usar **$_REQUEST**, se non importa o tipo de datos enviados desde o formulario.

```
/*Exemplo de variables de formulario máis complexas */
 <form action="array.php" method="post">
 Name: <input type="text" name="personal[name]"><br>
 Email: <input type="text" name="personal[email]"><br>
 Beer: <br>
 <select multiple name="beer[]">
        <option value="warthog">Warthog
        <option value="guinness">Guinness
        <option value="stuttgarter">Stuttgarter Schwabenbräu
 </select>
 <input type="submit">
 </form>
```

Cookies:

As cookies son un mecanismo para almacenar datos no navegador e así rastrexar ou identificar a usuarios que volven. Pódense crear cookies usando a función SetCookie(). As cookies son parte da cabeceira HTTP, así que se debe chamar á función SetCookie antes de que se envíe calquera saída ao navegador. É a mesma restrición que para a función header(). Calquera cookie que se reciba procedente do cliente será convertida automaticamente nunha variable de PHP como cos datos nos métodos GET e POST.

Se se queren asignar múltiples valores a unha soa cookie, basta con engadir [] ao nome da. Por exemplo:

```
SetCookie ("MyCookie[ ]", "Testing", estafe() 3600);
```

Nótese que unha cookie substituirá a unha cookie anterior que tivese o mesmo nome no navegador a menos que o camiño (path) ou o dominio fosen diferentes. Así, para unha aplicación de carro da compra poderíase querer manter un contador e ir pasándoo.

Exemplo:

```
$Count++ ;
SetCookie ("Count", $Count, time() + 3600);
SetCookie ("Cart[$Count]", $item, time() + 3600);
```

Sesións:

As sesións son unha forma sinxela de almacenar datos de usuarios de xeito individual usando un ID de sesión único. Isto pódese usar para facer persistente a información de estado entre peticións de páxinas. Os ID de sesións normalmente son enviados ao navegador mediante cookies de sesión, e o ID úsase para recuperar os datos de sesión existente. A ausencia de un ID o unha cookie de sesión permítelle saber a PHP para crear una nova sesión e xerar un novo ID de sesión.

As sesións seguen un fluxo de traballo sinxelo. Cando unha sesión se inicia, PHP recuperará unha sesión existente usando o ID pasado (normalmente dende unha cookie de sesión) ou, se non se pasa unha sesión, crearase unha sesión nova. PHP encherá a variable superglobal **$_SESSION** con calquera datos de sesión de que se inicie a sesión. Cando PHP pecha, automáticamente toma o contido d variable superglobal **$_SESSION** a serializa, e a envía para almacenala usando o xestor de almacenamento de sesións.

Por omisión, PHP usa o xestor interno de almacenamento **files**, o cal establécese mediante **session.save_handler**. Este garda os datos de sesión no servidor na localización especificada pola directiva de configuración **session.save_path**.

As sesións pódese iniciar manualmente usando a función **session_start()** se a directiva **session.auto_start** se establece a 1, unha sesión iniciarase automaticamente no momento en que PHP envíe calquera saída ao buffer de saída.

As sesións normalmente péchanse automaticamente cando PHP termina de executar un script, pero pódense pechar manualmente usando a función session_write_close().

```
/*Exemplo #1 Rexistrar unha variable con _SESSION. $ */
session_start();
if (!isset($_SESSION['count'])) {
        $_SESSION['count'] = 0;
} else {
        $_SESSION['count']                                                    ;
}
/*Exemplo #2 Deixar de rexistrar unha variable con _SESSION $ */
session_start();
unset($_SESSION['count']);
```

Conexión a bases de datos:

A partir da versión 4.1 MySQL incorpora a extensión **mysqli**, esta extensión permítenos o acceso á base de datos mediante funcións, como en versións anteriores, ou mediante obxectos. Antes de facela conexión e mellor crear un arquivo cos seguintes datos:

```
<?php
        $server = "localhost";
        $user = "nome_do_usuario";
        $pwd = "password";
        $bd = "nome_base_datos";
?>
```

que identifican ao servidor, usuario, crave e base da datos. Pódemos chamarlle “condat.inc.php” para os exemplos.

```
/*Exemplo de conexión con funcións
 <?php
        include("condat.inc.php");
        $dbh = mysql_connect($server, $user, $pwd);
        if(!$dbh) {
                echo ("Erro. Fallou a conexión á BBDD");
                exit();
        }
        $dbs = mysql_select_db($bd);
        if(!$dbs) {
                echo ("Erro. Non se puido acceder á BBDD. Inténteo máis tarde");
                exit();
        }

// Código de entrada de datos
        $fecha = date("Y-m-d H:i:s");
        $nombre = strtoupper( $_POST['nombre'] );
        $nombre = htmlspecialchars( $nombre );
        $qry = "INSERT INTO anuncios (nombre,fechaa)
                VALUES ('$nombre','$fecha')";
/*colle a data do sistema e dálle o formato para MySQL, colle o campo nome do formulario, $POST['nombre'], e convérteo a maiúsculas; strouppers, non deixa escribir caracteres HTML; htmlspecialchars e por último mete os datos na base de datos. */

  //Código de saída de datos
        $result = mysql_query("SELECT * FROM anuncios ORDER BY fechaa DESC");
        echo "<table align='center' border='1' bordercolor='#FF0000' width='25%'>\n";
        while( $row=mysql_fetch_array($result) )
        {
                echo "<tr>";
                echo "<td width='100%' align='left' >";
                $ano = substr( $row["fechaa"],0,4 );
                $mes = substr( $row["fechaa"],5,2 );
                $dia = substr( $row["fechaa"],8,2 );
                echo ( "<font color='#ff0000'>  Fecha   :".$dia."-".$mes."-".
                        $ano."</font>". "<br>" );
```

```
                echo ucwords( "<font color='#ff0000'>  Nombre  :".
                        $row["nombre"]."</font>". "<br>");
        echo"</td>\n";
        }
        echo "</table>\n";
?>
```

/* $result garda o contido da consulta. mysql_fetch_array e un array que contén as filas e que podemos percorrer con while. Table, tr e td son instrucións HTML para meter os datos que sacamos da base de datos nunha táboa. */

```
/* Exemplo de conexión con obxectos */
<?php
        include("condat.inc.php");
        $conect = new mysqli( $server, $user, $pwd, $bd ); // obxecto da conexión
        if( mysqli_connect_errno() ) {
                echo ("Erro na copnexión");
                exit();
        }

// Código de entrada de datos
        $fecha = date("Y-m-d H:i:s");
        $nombre = strtoupper( $_POST['nombre'] );
        $nombre = htmlspecialchars( $nombre );
        $entrar = "INSERT INTO anuncios (nombre,fechaa)
                        VALUES ('$nombre','$fecha')";

//Código de saída de datos
        $result = $conect->query("SELECT * FROM anuncios ORDER BY fechaa DESC");
        /* obxecto da consulta a b.d. */
        echo "<table align='center' border='1' bordercolor='#FF00FF' width='25%'>\n";
        while( $row = $result->fetch_array() )  //fetch_array contén o resultado da consulta
        {
                echo "<tr>";
                echo "<td width='100%' align='left' >";
                $ano = substr( $row["fechaa"],0,4 );
                $mes = substr( $row["fechaa"],5,2 );
                $dia = substr( $row["fechaa"],8,2 );
                echo ( "<font color='#ff0000'>  Fecha   :".$dia."-".$mes."-".
                        $ano."</font>". "<br>" );
                echo ucwords( "<font color='#ff0000'>  Nombre  :".
                        $row["nombre"]."</font>". "<br>");
                echo ( "<font color='#ff0000'>  Tel&eacute;fono:".
                        $row["telefo"]."</font>" . "<br>" );
                echo"</td>\n";
        }
        echo "</table>\n";
        $result->close;
        $conect->close;
```

```
?>
```

O uso da extensión mysqli proporciona dúas clases a hora de traballar con bases de datos MySQL: a clase mysqli e a clase mysqli_result.

Referencias:

Manual de PHP por Stig Sæther Bakken, Alexander Aulbach, Egon Schmid, Jim Winstead, Lars Torben Wilson, Rasmus Lerdorf, Andrei Zmievski, y Jouni Ahto
POO y MVC por Eugenia Bahit.