Contenido

MANUAL DE PHP

1.- Elementos del Lenguaje.

1.1.- Comentarios.

Para insertar comentarios en los scripts de PHP podemos optar entre varios métodos y varias posibilidades:
• Una sola línea
Basta colocar los símbolos // al comienzo de la línea o detrás del punto y coma que señala el final de una instrucción.
También se puede usar el símbolo # en cualquiera de las dos posiciones.
• Varias líneas
Si un comentario va a ocupar más de una línea podremos escribir /* al comienzo de la primera de ellas y */ al final de la última. Las líneas intermedias no requieren de ningún tipo de marca.
Los comentarios para los que usemos la forma /* ... */ no pueden anidarse.

1.2.- Constantes.

Una constante es un valor –un número o una cadena– que no va a ser modificado a lo largo del proceso de ejecución de los scripts que contiene un documento.
Para mayor comodidad, a cada uno de esos valores se le asigna un nombre, de modo que cuando vaya a ser utilizado baste con escribir su nombre.
Cuando ponemos nombre a una constante se dice que definimos esa constante.
En PHP las constantes se definen mediante la siguiente instrucción:
define("Nombre","Valor");

Los valores asignados a las constantes se mantienen en todo el documento, incluso cuando son invocadas desde una función.
No es necesario escribir entre comillas los valores de las constantes cuando se trata de constantes numéricas.
Si se realizan operaciones aritméticas con constantes tipo cadena, y su valor comienza por una letra, PHP les asigna valor cero.
Si una cadena empieza por uno o varios caracteres numéricos, al tratar de operarla aritméticamente PHP considerará únicamente el valor de los dígitos anteriores a la primera letra o carácter no numérico.
El punto entre caracteres numéricos es considerado como separador de parte decimal.

Ejemplo:

```
<?php
/* Definiremos la constante EurPta y le asignaremos el valor 166.386 */
define("EurPta",166.386);
 /* Definiremos la constante PtaEur asignándole el valor 1/166.386
En este caso el valor de la constante es el resultadode la operación aritmética
dividir 1 entre 166.386*/
define("PtaEur",1/166.386);
/* Definimos la constante Cadenas y le asignamos el valor:
        12Esta constante es una cadena*/
define("Cadena","12Esta constante es una cadena");
/* Definimos la constante Cadena2 y le asignamos el valor:
        12.54Constante con punto decimal*/
define("Cadena2","12.54Constante con punto decimal");
/* Comprobaremos los valores.
Si utilizamos echo enlazamos varias cadenas separadas conpunto y/o coma.
        Si utilizamos print enlazamos varias cadenas separadas con punto.*/
echo "Valor de la constante EurPta: ", EurPta, "<BR>";
echo "Valor de la constante PtaEur: ". PtaEur . "<BR>";
print "Valor de la constante Cadena: " . Cadena . "<BR>";
print "Valor de la constante Cadena x EurPta: " . Cadena*EurPta ."<br>";
print "Valor de la constante Cadena2 x EurPta: " . Cadena2*EurPta ."<br>";
echo "Con echo los números no necesitan ir entre comillas: " ,3,"<br>";
print "En el caso de print si son necesarias: " . "7" . "<br>";
print ("incluso entre paréntesis necesitan las comillas: "."45"."<br>");
print "Solo hay una excepción en el caso de print. ";
print "Si los números van en un print independiente no necesitan comillas ";
print 23;
?>
```

```
Valor de la constante EurPta: 166.386
Valor de la constante PtaEur: 0.0060101210438378
Valor de la constante Cadena: 12Esta constante es una cadena

Notice: A non well formed numeric value encountered
in C:\xampp\htdocs\PhpProject1<br>ewEmptyPHPWebPage4.php on line 32
Valor de la constante Cadena x EurPta: 1996.632

Notice: A non well formed numeric value encountered
in C:\xampp\htdocs\PhpProject1<br>ewEmptyPHPWebPage4.php on line 33
Valor de la constante Cadena2 x EurPta: 2086.48044
Con echo los números no necesitan ir entre comillas: 3
En el caso de print si son necesarias: 7
incluso entre paréntesis necesitan las comillas: 45
Solo hay una excepción en el caso de print. Si los números van en un print independiente no
necesitan comillas 23
```

1.3.- Funciones de salida.

echo

La función echo, aunque admite también la forma echo(), no requiere de forma obligatoria el uso de los paréntesis.
Detrás de la instrucción echo pueden insertarse: variables, cadenas (éstas entre comillas) y números (éstos sin comillas) separadas normalmente por comas.
Este es un ejemplo de código:

```
$a=24; $b="Pepe";
$c="<br>";
echo $a,$b,25,
"Luis",$c;
```

que produciría esta salida:
24Pepe25Luis

Observa los valores que hay detrás de echo, no es necesario insertar todos los valores en la misma línea.

print()

La función print() sólo puede contener dentro del paréntesis una sola variable, o el conjunto de varias de ellas enlazadas (concatenadas) por un punto. En algunas versiones los paréntesis no son necesarios.

Algunos ejemplos:

```
print(25.3)
```
produciría esta salida
25.3

```
print("Gonzalo")
```
escribiría
Gonzalo

```
$z=3.1416;
print($z);
```
escribiría
3.1416

Es posible utilizar dentro del paréntesis el concatenador de cadenas.

```
$h=3;
$f=" hermanos"
print("Heladería ".$h.$f)
```
saldría:
Heladería 3hermanos

1.4.-Variables.

Los nombres de variables comienzan con el signo $ y son sensibles a mayúsculas y minúsculas (no así las palabras claves del lenguaje).

En PHP no es necesario definir el tipo antes de utilizarla, las mismas se crean en el momento de emplearlas. Por esto se considera a PHP un "lenguaje no tipado". Este tipo de lenguajes son muy intuitivos y fáciles de programar pero generan una serie de inconvenientes que pueden restarle potencia.

Las variables se declaran cuando se le asigna un valor, por ejemplo:

```
$dia = 24;              //Se declara una variable de tipo integer.
$sueldo = 758.43;       //Se declara una variable de tipo double.
$nombre = "juan";       //Se declara una variable de tipo string.
$existe = true;         //Se declara una variable boolean.
```

Para imprimir variables normalmente utilizamos el comando echo.
Un programa completo que inicializa y muestra el contenido de cuatro variables de distinto tipo es:

```
<?php
        $dia = 24;
        $sueldo = 758.43;
        $nombre = "juan";
        $existe = true;
        echo "Variable entera:";
        echo $dia;
        echo "<br>";
        echo "Variable double:";
        echo $sueldo;
        echo "<br>";
        echo "Variable string:";
        echo $nombre;
        echo "<br>";
        echo "Variable boolean:";
        echo $existe;
?>
```

```
Variable entera:24
Variable double:758.43
Variable string:juan
Variable boolean:1 // con valor false no saca nada.
```

Hemos utilizado echo para mostrar los mensajes, contenido de variables y finalmente para imprimir marcas HTML.
Toda variable tiene un ámbito de uso que corresponde con el script en el que está definida. Esto cambia si se utilizan funciones en dicho script. Entonces las variables pertenecen al ámbito de la función en el que están inscritas.

Variables globales

Las funciones pueden utilizar valores de *variables externas a ellas* pero ello requiere incluir *dentro de la propia función* la siguiente instrucción:

global nombre de la variable**;**
Por ejemplo: **global $a1;**

En una instrucción –**global**– pueden definirse como tales, de forma simultánea, varias variables. Basta con escribir los nombres de cada una de ellas separados por comas.

P. ej.: global $a1, $a2, $a3;

Variables superglobales

A partir de la versión 4.1.0de PHP se ha creado un nuevo tipo de variables capaces de comportarse como globales sin necesidad de que se definan como tales.
Estas variables que no pueden ser creadas por usuario, recogen de forma automática información muy específica y tienen nombres preasignados que no pueden modificarse.
Estas variables están en constante cambio dependiendo de la versión de PHP.

Algunas de ellas:
$_SERVER, **$_POST**, **$_GET,$_ENV o $_REQUEST**

Ejemplo:

```
<?php
        echo $_SERVER['HTTP_HOST'],"<BR>";
       echo $_SERVER['HTTP_USER_AGENT'],"<BR>";
        echo $_SERVER['HTTP_ACCEPT'],"<BR>";
        echo $_SERVER['HTTP_ACCEPT_LANGUAGE'] ,"<BR>";
        echo $_SERVER['HTTP_ACCEPT_ENCODING'],"<BR>";
        echo $_SERVER['HTTP_ACCEPT_CHARSET'],"<BR>";
              # No funciona en PHP 5.4
        echo $_SERVER['HTTP_CONNECTION'],"<BR>";
       echo $_SERVER['PATH'],"<BR>";
?>
```

Ejemplo de uso de variables y ámbito:

```
<?php
       # Definimos la variable $carlos como vacía
       $carlos="";
       # Definimos las variables $Carlos y $Alicia (ojo con mayúsculas y minúsculas)
       $Carlos="Me llamo Carlos";
       $Alicia="Me llamo Alicia";
?>
<!-- esto es HTML, hemos cerrado el script -->
<center><b>Vamos a ver el contenido de las variables</b></center>
<!--   un nuevo script PHP -->

<?php
       echo "<br> El valor de la variable carlos es: ",$carlos;
       echo "<br> No ha puesto nada porque $carlos esta vacía";
       echo "<br> El valor de la variable Carlos es: ",$Carlos;
?>
```

```
<center><b><br>Invocando la variable desde una función</b></center>

<?php
/* Escribiremos una function llamada vervariable*/
/*Mantenemos las definiciones de variables del ejemplo anterior*/
function vervariable(){
        echo "<br> Si invoco la variable Carlos desde una función";
        echo "<br>me aparecerá en blanco";
        echo "<br>me aparecerá en blanco: $Carlos  _____";
        echo "<br>El valor de la variable Carlos es: ",$Carlos;
}
/* Haremos una llamada a la funcion vervariable.*/
vervariable();
?>

<!-- mas HTML puro  -->
<center><b><br>Ver la variable desde la función
poniendo <i>global</i></b></center>

<?php
# una nueva funcion
function ahorasi(){
        # aqui definiremos a $Carlos como global
        # la función leerá su valor externo
        global $Carlos;
        echo "<br><br> Hemos asignado ámbito global a la variable";
        echo "<br>ahora Carlos aparecerá";
        echo "<br>El valor de la variable  Carlos es: ", $Carlos;
}
# Tendremos que invocarla para que se ejecute ahora
ahorasi();
?>

<center><b><br>Un solo nombre y dos <i>variables distintas</i></b><br>
Dentro de la función el valor de la variable es <br></center>

<?php
function cambiaAlicia(){
        $Alicia="Ahora voy a llamarme Elena";
        echo "<br>",$Alicia;
        /* Sugerencia:
        Probar con global $Alicia="Ahora voy a llamarme Elena";
        Observar la salida.*/
}

cambiaAlicia();
?>
<center>... pero después de salir de la funciónvuelvo al valor original...</center>
<?php
echo "<br>",$Alicia;
?>
```

Salida:

Vamos a ver el contenido de las variables

El valor de la variable carlos es:

```
No ha puesto nada porque esta vacía
El valor de la variable Carlos es: Me llamo Carlos

Invocando la variable desde una función

Si invoco la variable Carlos desde una función
me aparecerá en blanco
Notice: Undefined variable: Carlos
in C:\xampp\htdocs\PhpProject1<br>ewEmptyPHPWebPage6.php on line 41

me aparecerá en blanco: _____
El valor de la variable Carlos es:
Notice: Undefined variable: Carlos
in C:\xampp\htdocs\PhpProject1<br>ewEmptyPHPWebPage6.php on line 42

Ver la variable desde la función poniendo global

Hemos asignado ámbito global a la variable
ahora Carlos aparecerá
El valor de la variable Carlos es: Me llamo Carlos

Un solo nombre y dos variables distintas
Dentro de la función el valor de la variable es

Ahora voy a llamarme Elena
... pero después de salir de la función vuelvo al valor original...

Me llamo Alicia
```

Variables estáticas.

Para poder conservar el último valor de una variable definida *dentro de una función* basta con definirla como **estática**.

La instrucción que permite establecer una variable como **estática** es la siguiente:

static nombre = valor;

P. ej: si la variable fuera **$a** y el **valor inicial** asignado fuera **3** escribiríamos:

```
static $a=3;
```

La variable conservará el último de los valores que pudo habérsele asignado durante la ejecución de la función que la contiene. No retomará el valor inicial hasta que se actualice la página.

Ejemplo:

```
<?php
        function sinEstaticas(){
              # Pongamos aquí sus valores iniciales
               $a=0;
               $b=0;
              # Imprimamos estos valores iniciales
               echo "Valor inicial de",' $a: ',$a,"<br>";
               echo "Valor inicial de",' $b: ',$b,"<br>";
               $a +=5;
               $b -=7;
              # Visualicemos los nuevos valores de las variables
               echo "Nuevo valor de",' $a: ',$a,"<br>";
               echo "Nuevo valor de",' $b: ',$b,"<br>";
       }
```

```
        function conEstaticas(){
                # Definimos $b como estática
                 $a=0;
                 static $b=0;
                 echo 'Valor inicial de  $a: ',$a,"<br>";
                 echo "Valor inicial de”,’ $b: ‘,$b,"<br>";
                 $a +=5;
                 $b -=7;
                 echo "Nuevo valor de”,’ $a: ‘,$a,"<br>";
                 echo "Nuevo valor de”,’ $b: ‘,$b,"<br>";
        }

 print ("Esta es la primera llamada a sinEstaticas()<br>");
 # Invoquemos la función sinEstaticas;
 sinEstaticas();
 # Añadamos un nuevo comentario a la salida
 print ("Esta es la segunda llamada sinEstaticas()<br>");
 print ("Debe dar el mismo resultado que la llamada anterior<br>");
 # Invoquemos por segunda vez sinEstaticas;
 sinEstaticas();
 # Hagamos ahora lo mismo con la función conEstaticas
 print ("Esta es la primera llamada a conEstaticas()<br>");
 conEstaticas();
 print ("Esta es la segunda llamada a conEstaticas()<br>");
 print ("El resultado es distinto a la llamada anterior<br>");
 conEstaticas();
 ?>
```

Salida:

```
Esta es la primera llamada a sinEstaticas()
Valor inicial de $a: 0
Valor inicial de $b: 0
Nuevo valor de $a: 5
Nuevo valor de $b: -7
Esta es la segunda llamada sinEstaticas()
Debe dar el mismo resultado que la llamada anterior
Valor inicial de $a: 0
Valor inicial de $b: 0
Nuevo valor de $a: 5
Nuevo valor de $b: -7
Esta es la primera llamada a conEstaticas()
Valor inicial de $a: 0
Valor inicial de $b: 0
Nuevo valor de $a: 5
Nuevo valor de $b: -7
Esta es la segunda llamada a conEstaticas()
El resultado es distinto a la llamada anterior
Valor inicial de $a: 0
Valor inicial de $b: -7
Nuevo valor de $a: 5
Nuevo valor de $b: -14
```

Variables de variables.

Además del método habitual de asignación de nombres a las variables -poner el signo $ delante de una palabra-, existe la posibilidad de que tomen como nombre el valor de otra variable previamente definida.
La forma de hacerlo sería esta:
$$nombre_variable_previa;

Ejemplo.
Supongamos que tenemos una variable como esta:
$color="verde";
Si ahora queremos definir una nueva variable que utilice como nombre el valor (verde) que está contenido en la variable previa ($color), habríamos de poner algo como esto:
$$color="no me gusta";

¿Cómo podríamos visualizar el valor de esta nueva variable?
Habría tres formas de escribir la instrucción:
print $$color;
o
print ${$color};
o también
print $verde;

Cualquiera de las instrucciones anteriores nos produciría la misma salida: “no me gusta”.
Cuando la variable utilizada para definir una variable de variable cambia de valor no se modifica ni el nombre de esta última ni tampoco su valor.

Ejemplo:
```
<?php
# Definamos una variable y asignémosle un valor
$color="rojo";
# Definamos ahora una nueva variable de nombre variable
# usando para ello la variable anterior
$$color=" es mi color preferido";
# Veamos impresos los contenidos de esas variables
print ( "El color ".$color. $$color ."<br>");
#o también
print ( "El color ".$color. ${$color}."<br>");
# o también
print ( "El color ".$color. $rojo."<br>");
print ("Las tres líneas anteriores deben decir lo mismo<br>");
print ("Hemos invocado la misma variable de tres formas diferentes<BR>");
# cambiemos ahora el nombre del color
$color="magenta";
print ("Ahora la variable $color ha cambiado a magenta<br>");
print ("pero como no hemos creado ninguna variable con ese color<br>");
print ("en las lineas siguientes no aparecerá nada <br>");
print ("detrás de la palabra magenta <br>");

# ¡OjO!
print (" El color ".$color.$$color."<br>");
print (" El color ".$color.${$color}."<br>");
# Comprobemos que la variable $rojo creada como variable de variable
# cuando $color="rojo" aún existe y mantiene aquel valor
print ("Pese a que $color vale ahora ".$color."<br>");
print ("la vieja variable $rojo sigue existiendo <br>");
print ("y conserva su valor. Es este: ".$rojo);
?>
```

Salida:

```
El color rojo es mi color preferido
El color rojo es mi color preferido
El color rojo es mi color preferido
Las tres líneas anteriores deben decir lo mismo
Hemos invocado la misma variable de tres formas diferentes
Ahora la variable magenta ha cambiado a magenta
pero como no hemos creado ninguna variable con ese color
en las lineas siguientes no aparecerá nada
detrás de la palabra magenta
Notice: Undefined variable: magenta in
 C:\xampp\htdocs\PhpProjectBasico<br>ewEmptyPHPWebPage.php
on line 34
El color magenta
Notice: Undefined variable: magenta in
 C:\xampp\htdocs\PhpProjectBasico<br>ewEmptyPHPWebPage.php
on line 35
El color magenta
Pese a que magenta vale ahora magenta
la vieja variable es mi color preferido sigue existiendo
y conserva su valor. Es este: es mi color preferido
```

Variables de tipo String.

Una variable de este tipo puede almacenar una serie de caracteres.

```
$cadena1="Hola";
$cadena2="Mundo";
echo $cadena1." ".$cadena2;
```

Para concatenar string empleamos el operador<**.**>
Tengamos en cuenta que el comando echo de más arriba lo podemos hacer más largo de la siguiente forma:

```
echo $cadena1;
echo " ";
echo $cadena2;
```

Cuando una cadena encerrada entre comillas dobles contiene una variable en su interior, ésta se trata como tal, por lo tanto se utilizará su contenido para el almacenamiento.

```
$dia=10;
$fecha="Hoy es $dia";
echo $fecha;
```

En pantalla se muestra: Hoy es 10.

Es decir, en la cadena, se sustituye el nombre de la variable $dia, con el contenido de la misma.
Una cadena se puede definir con las comillas simples (pero es importante tener en cuenta que no se sustituyen las variables si empleamos comillas simples):

```
$nombre='juan carlos';
```

Operaciones con cadenas.

La **concatenación** de cadenas
Para concatenar (unir), en una sola, varias porciones de texto hemos venido utilizando –en las instrucciones print y echo– un punto (.).

El **operador .**
Este punto es un elemento muy importante que, además de la forma que hemos visto en las páginas anteriores, tiene los siguientes usos:
Unir dos cadenas y recogerlas en una variable
Con la sintaxis:

```
$a="cad1" . "cad2";
```

o mediante

```
$a= $b . $c;
```

podemos obtener una nueva variable formada por la unión dos trozos.

Sea cual sea el contenido de las variables concatenadas, serán tratadas por PHP como de tipo cadena y la variable que contiene el resultado es del tipo string.

Añadir contenidos a una variable tipo string.
Si utilizamos una sintaxis como esta:

```
$a .="cad1";
```

o de este otro tipo

```
$a .=$b;
```

se añadiría al valor actual de la variable $a el contenido indicado después del signo igual.
Fíjate en la importancia del punto. Si está presente, se añaden nuevos contenidos a la variable; pero en el caso de que no lo estuviera, se sustituirían esos contenidos, con lo cual se perdería la información que esa variable pudiera contener.

Tratamiento de cadenas.

Funciones de cadenas
Algunas de las funciones que permiten manejar los formatos de las cadenas de caracteres son estas:

chr(n)
Devuelve el carácter cuyo código ASCII es n.

ord(cadena)
Devuelve el código ASCII del primero de los caracteres de la cadena.

strlen(cadena)
Devuelve la longitud (número de caracteres) de la cadena. Los espacios son considerados como un carácter más.

strtolower(cadena)
Cambia todos los caracteres de la cadena a minúsculas.

strtoupper(cadena)
Convierte en mayúsculas todos los caracteres de la cadena.

ucwords(cadena)
Convierte a mayúsculas la primera letra de cada palabra.

ucfirst(cadena)
Convierte a mayúsculas la primera letra de la cadena y pone en minúsculas todas las demás.

ltrim(cadena)
Elimina todos los espacios que pudiera haber al principio de la cadena.

rtrim(cadena)
Elimina todos los espacios que existieran al final de la cadena.

trim(cadena)
Elimina los espacios tanto al principio como al final de la cadena

chop(cadena)
Elimina los espacios al final de la cadena. Es idéntica a rtrim.

Tanto trim, como ltrim y rtrim eliminan, además de los espacios,
las secuencias:
, \r, \t, \v y \0, llamadas también caracteres protegidos.

preg_replace(cadena)
Elimina los espacios dentro de la cadena

```
$cadenaPruebasinEspacios=preg_replace("[\s+]","",$cadenaPrueba);
echo "Cadena sin espacios:", $cadenaPruebasinEspacios,"<br>";
```

substr(cadena,n)
Si el valor de n es positivo extrae todos los caracteres de la cadena a partir del que ocupa la posición enésima a contar desde la izquierda.
Si el valor de n es negativo serán extraídos los n últimos caracteres contenidos en la cadena.

substr(cadena,n,m)
Si n y m son positivos extrae m caracteres a partir del que ocupa la posición enésima, de izquierda a derecha.
Si n es negativo y m es positivo extrae m (contados de izquierda a derecha) a partir del que ocupa la posición enésima contada de derecha a izquierda.
Si n es positivo y m es negativo extrae la cadena comprendida entre el enésimo carácter (contados de izquierda a derecha) hasta el emésimo, contando en este caso de derecha a izquierda.
Si n es negativo y m también es negativo extrae la porción de cadena comprendida entre el emésimo y el enésimo caracteres contando, en ambos casos, de derecha a izquierda. Si el valor absoluto de n es menor que el de m devuelve una cadena vacía.

strrev(cadena)
Devuelve la cadena invertida

str_repeat(cadena, n)
Devuelve la cadena repetida tantas veces como indica n.

str_pad(cad, n, rell, tipo)

Añade a la cadena cad los caracteres especificados en rell (uno o varios, escritos entre comillas)hasta que alcance la longitud que indica n (un número)
El parámetro tipo puede tomar uno de estos tres valores (sin comillas):
STR_PAD_BOTH (rellena por ambos lados)
STR_PAD_RIGHT (rellena por la derecha)
STR_PAD_LEFT (rellena por la izquierda).
Si se omite la cadena de Relleno utilizará espacios y si se omite el tipo rellenará por la derecha.

Ejemplo:

```
<?php
        #definamos y asignemos valores a variables tipo cadena
         $cadena1="Esto es una cadena de texto";
         $cadena2="Esta es una segunda cadena de texto";
        #hagamos lo mismo con variables numéricas
         $cadena3=127;
         $cadena4=257.89;
        # unámoslas mezclando tipos
         $union1=$cadena1 . $cadena2;
         $union2=$cadena1 . $cadena3;
         $union3=$cadena3 . $cadena4;
        #veamos qué ha ocurrido
         echo $union1,"<br>";
         echo $union2,"<br>";
         echo $union3,"<br>";
        # modifiquemos ahora una cadena añadiéndole contenidos
         $cadena3 .=" Este es el texto que se añadirá a la variable cadena3";
        # imprimamos los resultados
         echo $cadena3,"<br>";
      ?>
```

Salida:

```
Esto es una cadena de textoEsta es una segunda cadena de texto
Esto es una cadena de texto127
127257.89
127 Este es el texto que se añadirá a la variable cadena3
```

Operaciones aritméticas.

Operadores aritméticos

Suma
$a + $b

Diferencia
$a - $b

Producto
$a * $b

Cociente
$a / $b

Cociente entero
(int)($a / $b)

Resto de la división
$a % $b

Raíz cuadrada
sqrt($a)

Potencia a^b
pow($a,$b)

Operadores de comparación.

PHP dispone de los siguientes operadores de comparación:

$A == $B
El operador == compara los valores de dos variables y devuelve 1 (CIERTO) en el caso de que sean iguales y el valor NULL –carácter ASCII 0– (FALSO) cuando son distintas.
Mediante este operador se pueden comparar variables de distinto tipo.
Para comparar una cadena con un número se extrae el valor entero de la cadena (si lleva dígitos al comienzo los extrae y en caso contrario le asigna el valor cero) y utiliza ese valor para hacer la comparación.
Cuando se comparan cadenas discrimina entre mayúsculas y minúsculas ya que utiliza los códigos ASCII de cada uno de los caracteres para hacer la comparación.
La comparación se hace de izquierda a derecha y devuelve 1 (CIERTO) sólo en el caso que coincidan exactamente los contenidos de ambas cadenas.

$A === $B
El operador === es similar al anterior, pero realiza la comparación en sentido estricto.
Para que devuelva 1 es necesario que sean iguales los valores de las variables y también su tipo.

$A != $B
El operador **!=** devuelve 1 cuando los valores de las variables son distintos (en general ! indica negación, en este caso podríamos leer «no igual») y devuelve NUL cuando son iguales.
Este operador no compara en sentido estricto, por lo que puede considerar iguales los valores de dos variables de distinto tipo.

$A < $B
El operador < devuelve 1 cuando los valores de $A son menores que los de $B.

Los criterios de comparación son los siguientes:
– Los valores numéricos siguen el criterio matemático.
– Cuando se trata de un número y una cadena extrae el valor numérico de ésta (es cero si no hay ningún dígito al principio de la misma) y hace una comparación matemática.

– En el supuesto de dos cadenas, compara uno a uno –de izquierda a derecha– los códigos ASCII de cada uno de los caracteres (primero con primero, segundo con segundo, etcétera).
Si al hacer esta comprobación encuentra –en la primera cadena– un carácter cuyo código ASCII es mayor que el correspondiente de la segunda cadena, o encuentra que todos son iguales en ambas cadenas devuelve NUL. Solo en el caso de no existir ninguno mayor y sí haber al menos uno menor devolverá UNO.
– Cuando las cadenas tengan distinta longitud, considerará (a efectos de la comparación) que los caracteres que faltan en la cadena más corta son NULL (ASCII 0).

$A <= $B
Se comporta de forma idéntica al anterior. La única diferencia es que ahora aceptará como ciertos los casos de igualdad tanto en el caso de números como en el de códigos ASCII.

$A > $B
Es idéntico –en el modo de funcionamiento– a $A < $B. Sólo difiere de éste en el criterio de comparación que ahora requerirá que los valores de $A sean mayores que los de la variable $B.

$A >= $B
Añade al anterior la posibilidad de certeza en caso de igualdad.

Operadores lógicos

Mediante operadores lógicos es posible evaluar un conjunto de variables lógicas, es decir, aquellas cuyos valores sean únicamente:
VERDADERO o FALSO (1 ó NULL).
El resultado de esa evaluación será siempre 1 ó NULL.

$A AND $B
El operador AND devuelve VERDADERO (1) en el caso de que todas las variables lógicas comparadas sean verdaderas, y FALSO (NULL) cuando alguna de ellas sea falsa.

$A && $B
El operador && se comporta deforma idéntica al operador AND. La única diferencia entre ambos es que operan con distinta precedencia.
Más abajo veremos el orden de precedencia de los distintos operadores.

$A OR $B
Para que el operador OR devuelva VERDADERO (1) es suficiente que una sola de las variables lógicas comparadas sea verdadera.
Únicamente devolverá FALSO (NULL) cuando todas ellas sean FALSAS.

$A || $B
El operador || se comporta de forma idéntica al operador OR.
Su única diferencia es el orden de precedencia con el que opera.

$A XOR $B
El operador XOR devuelve VERDADERO (1) sólo en el caso duque sea cierta una sola de las variables, y FALSO (NULL) cuando ambas sean ciertas o ambas sean falsas.

! $A
Este operador NOT (negación) devuelve VERDADERO (1) si la variable lógica $A es FALSA y devuelve FALSO (NUL) si el valor de esa variable $A es VERDADERO.

Sintaxis alternativa
Tal como hemos descrito los distintos operadores lógicos sería necesario que $A y $B contuvieran valores lógicos, y eso requeriría un paso previo para asignarles valores de ese tipo.
Habría que recurrir a procesos de este tipo:
$A = $x>$y;
$B= $x >=$z;
$A && $B;
pero se obtendría el mismo resultado escribiendo:
$x>$y && $x >=$z;
que, aparte de ser la forma habitual de hacerlo, nos evita dos líneas de instrucciones.
Aunque el propio ejemplo se autocomenta, digamos que al utilizar operadores lógicos se pueden sustituir las variables lógicas por expresiones que den como resultado ese tipo de valores.

Orden de precedencia
Cuando se usan los operadores lógicos se plantean situaciones similares a lo que ocurre con las operaciones aritméticas.
Dado que permiten trabajar consecuencias de operaciones sería posible, por ejemplo, una operación de este tipo:
$a<$b **OR** $c<$b **&&** $a<3

Surgirían preguntas con estas:
– ¿qué comparación se haría primero OR o&&?
– ¿se harían las comparaciones En el orden natural?
– ¿alguno de los operadores tiene prioridad sobre el otro?
Igual que en las matemáticas, también aquí, hay un orden de prioridad que es el siguiente:

NOT, &&, ||, AND, XOR y, por último, OR.
De esta forma la operación && serializaría antes que ||, mientras que si pusiéramos AND en vez de &&sería la operación || la que se haría antes y, por lo tanto, los resultados podrían variar de un caso a otro.
Aquí también es posible, de la misma manera que en la aritmética, utilizar paréntesis para priorizar una operación frente a otra.

Ejemplo:

```
<?php
# asignemos valores a cuatro variables
$a=3;
$b=6;
$c=9;
$d=17;

# utilicemos operadores de comparación
# y recojamos sus resultados en nuevas variables
$x= $a<$b;
$y= $a<$b;
$z=$c>$b;

# apliquemos un operador lógico (por ejemplo &&)
# e imprimamos el resultado
print("Resultado FALSO si no sale nada: ".($y && $z)."<br>");


# hagamos la misma comparación sin utilizar la variables $y y $z
# que deberá darnos el mismo resultado
print("<br>Resultado FALSO si no sale nada: ".($a<$b && $c>$b)."<br>");
if($a<$b OR $c>$b && $d<$a) {
		print "cierto<br>";
	}else{
		print "falso<br>";
	}
if($a<$b || $c>$b AND $d<$a) {
		print "cierto<br>";
	}else{
		print "falso<br>";
	}

if($a<$b || ($c>$b AND $d<$a)) {
		print "cierto<br>";
	}else{
		print "falso<br>";
	}
?>
```

Salida:

```
Resultado FALSO si no sale nada: 1

Resultado FALSO si no sale nada: 1
cierto
falso
cierto
```

Operadores de incremento.

Los caracteres ++ y -- escritos al lado del nombre de una variable producen incrementos o decrementos de una unidad en el valor de la misma.
De igual forma, los operadores +=n y -=n escritos a la derecha del nombre de una variable producen incrementos o decrementos de n unidades en el valor de la variable.

Como veremos a continuación, los operadores ++ y -- se comportan de distinta forma según estén situados a la izquierda o a la derecha de la variable.
Estas operaciones sólo tienen sentido en variables numéricas –enteras o no–

Operadores de preincremento
++$A y --$A
Este operador incrementa el valor de la variable en una unidad (+1 o-1) antes de ejecutar el contenido dela instrucción.
$A+=n y $A-=n
Este operador incrementa el valor de la variable en n unidades (+n o-n) y luego ejecuta el contenido de la instrucción.

Operadores de post-incremento
$A++ y $A--
Cuando los operadores ++ o – están situados a la derecha de la variable
los incrementos no se producen hasta que se ejecute la instrucción siguiente.

2.- Estructuras de programación.

2.1.- Estructura condicional.

Los operadores condicionales son la herramienta que permite tomar decisiones tales como hacer o no hacer, y también hacer algo bajo determinadas condiciones y otra cosa distinta en caso de que no se cumplan.
La forma más simple es:

```
if(condición)
        ..instrucción... ;
```

Si se cumple la condición establecida en el paréntesis se ejecutará la primera instrucción que se incluya a continuación de ella.
Cualquier otra instrucción que hubiera a continuación de esa primera no estaría afectada por el condicional y se ejecutaría en cualquier circunstancia.
Cuando es necesario que –en caso de que se cumpla la condición o condiciones– se ejecute más de una instrucción, se añade una { para indicar que habrá varias instrucciones, se escriben estas y mediante } se señala el final.

```
if(condición){
        ..instrucción 1... ;
        ..instrucción 2... ;
        .... ;
}
```

PHP permite la utilización del operador condicional if con esta sintaxis. Una primer script PHP establece la condición. Todo lo contenido entre ese primer script y el de cierre: <? } ?> será código HTML (está fuera del script), que se insertará en el documento sólo en el caso de que se cumpla la condición.

```
<?php
....
if(condicion){
 ?>
        ..Etiquetas HTML... ;
        ..HTML... ;
        .... ;
<?php
 }
 ?>
```

if ... else
El operador condicional tiene una interesante ampliación. En conjunción con else permite añadir instrucciones que sólo serían ejecutadas en caso de no cumplirse la condición.

Esta nueva opción se habilita mediante la siguiente sintaxis:

```
if(condicion){
        ... instrucciones...
        ... a ejecutar cuando se cumple la condición
} else {
        ... instrucciones...
        ... a ejecutar cuando NO se cumple la condición
}
```

En algunos casos resulta útil y cómodo el uso de esta otra posibilidad de sintaxis:

```
(condición) ? (opc1) : (opc2);
```

Si se cumple la condición se ejecuta la opc1, pero en el caso de que no se cumpla se ejecutará la opc2.

if ... elseif .. else

Otra posibilidad dentro de la estructura de los operadores condicionales es la inclusión de elseif.

```
if(condicion1){
        ... instrucciones..... a ejecutar cuando se cumple la condición1
}elseif(condicion2){
        ... instrucciones..... a ejecutar cuando se cumple la condición2
        sin cumplirse condición1
} else {
        ... instrucciones..... a ejecutar cuando no se cumple ni la condición1
        ni la condicion2
}
```

Condicionales anidados

```
if(condición1){
        ... instrucciones...
        if(condición2){
                ... instrucciones...
        } else {
                ...instrucciones
        }
}else{
        ... instrucciones...
        ...instrucciones...
}
```

La función exit()

Cuando se ejecuta exit() se interrumpe la ejecución del script con lo que la respuesta del servidor ala petición del cliente incluirá únicamente los contenidos generados antes de su ejecución.

Ejemplo:

```
<?php
$A=3; $B="3";
if ($A==$B)
        print ("A es igual B");
if ($A<$B)
        print ("A es menor que B");
print("<br>A no es menor que b, pero esto saldrá<br>");
print("Esta es la segunda instrucción. No la condicionará el if");
$A=3; $B="3";
if ($A==$B){
        print ("A es igual B");
        echo "<br>";
        echo "Este if tiene varias instrucciones contenidas entre llaves";
}
$A=3; $B="4";
if ($A==$B){
        print ("A es igual B");
        echo "<br>";
        echo "Este if tiene varias instrucciones";
}else{
        print("A no es igual que B");
        echo "<br>";
        echo ("La estructura de control se ha desviado al else");
}
?>
```

Salida:

```
A es igual B
A no es menor que b, pero esto saldrá
Esta es la segunda instrucción. No la condicionará el if A es igual B
Este if tiene varias instrucciones contenidas entre llaves
A no es igual que B
La estructura de control se ha desviado al else
```

```
<?php
        $a=3;
        if ($a==5){
?>
      <H1>Esto no es PHP. A es igual 5</H1>
 <?php
        }else{
 ?>
      <H2>Esto no es PHP. Es el resultado del ELSE</H2>
 <?php }; ?>
 <?php
        $a=5;
        ($a==8) ? ($B="El valor de a es 8"): ($B="El valor de a no es 8");
          echo $B;
?>
```

Salida:

```
Esto no es PHP. Es el resultado del ELSE
El valor de a no es 8
```

La estructura switch
Una alternativa al uso de condicionales del tipo if es la función switch.
Se trata de un condicional que evalúa una variable y, según su valor, ejecuta unas instrucciones u otras.

La forma más habitual de uso de esta estructura es ésta:

```
switch ( variable ) {
        case n1:
                instrucciones caso n1...
                .....................
        break;
        case n2:
                instrucciones caso n2...
                .....................
        break;
}
```

Cuando se usa esta sintaxis solo se ejecutan aquellas instrucciones que han sido incluidas a partir de la etiqueta en la que el número que sigue a case coincide con el valor dela variable.

Esta opción incluye antes de cada nuevo case la instrucción de ruptura break. Cuando PHP encuentra el break interrumpe la ejecución y no la reanuda hasta la instrucción siguiente a la } que cierra la función switch.
Insertando break en cada una de las opciones case solo se ejecutarían las instrucciones contenidas entre case num y ese break.

default:
Bajo la clausula **default** se pueden incluir –dentro de la función switch– un conjunto de instrucciones que solo serán ejecutadas en el caso que el valor de la variable no coincida con ninguno de los case.

Ejemplo:

```
<?php
         $i=3;
         switch ($i) {
         case 0:
                print "La variable i es 0<br>";
                 break;
         case 1:
                print "La variable i es 1<br>";
                 break;
         case 2:
                print "La variable i es 2<br>";
                break;
        default:
                print "La variable i es mayor que dos o menor que cero";
                break;
          }
      ?>
```

2.2.- Estructuras repetitivas.

Estructura for.

```
for([Inicialización variable];[Condición];[Incr. o decrem. variable]) {
        [Instrucciones];
}
```

Ejemplo. Números del 1 al 100:

```
<?php
for($f=1;$f<=100;$f++)
{
        echo $f;
        echo "<br>";
}
?>
```

Estructura while.

```
while (condición){
      [Instrucciones];
};
```

Esta estructura está en casi todos los lenguajes. El bloque se repite mientras la condición del while sea verdadera.
La condición del while se verifica antes de entrar al bloque a repetir. Si la misma se verifica falsa la primera vez no se ejecutará el bloque. Veamos un ejemplo: Generar un valor aleatorio entre 1 y 100, luego imprimir en la página desde 1 hasta el valor generado (de uno en uno):

```
<?php
$valor=rand(1,100);
$inicio=1;
while($inicio<=$valor)
{
  echo $inicio;
  echo "<br>";
  $inicio++;
}
?>
```

Estructura do ... while.

Por último tenemos una estructura repetitiva similar al while llamada do/while, donde la condición se verifica después de ejecutarse el bloque repetitivo.

```
do {
        [Instrucciones];
} while (condición);
```

Ejemplo completo:

```
<?php
$filas=5; $columnas=3;
 print ("<table border=2 width=400 align=center>");
while ($filas>0):
echo "<tr>";
$filas--;
while ($columnas>0):
         echo "<td>";
          print "fila: ".$filas." columna: ".$columnas;
          print ("</td>");
          $columnas--;
endwhile;
$columnas=3;
echo "</TR>";
endwhile;
print "</table>";

$A=500;
do {
if ($A>=500) {
        echo "No puede ejecutarse el bucle, porque no se cumple la condicion";
break;
}
++$A;
echo "Valores de A usando el do: ",$A,"<br>";
} while($A<500);
echo "<BR>He salido del bucle porque A es: ",$A,"<BR>";

for ($i = 1; $i <= 10; $i++) {
print $i."<br>";
}
?>
```

fila: 4 columna: 3	fila: 4 columna: 2	fila: 4 columna: 1
fila: 3 columna: 3	fila: 3 columna: 2	fila: 3 columna: 1
fila: 2 columna: 3	fila: 2 columna: 2	fila: 2 columna: 1
fila: 1 columna: 3	fila: 1 columna: 2	fila: 1 columna: 1
fila: 0 columna: 3	fila: 0 columna: 2	fila: 0 columna: 1

No puede ejecutarse el bucle, porque no se cumple la condicion
He salido del bucle porque A es: 500
1
2
3
4
5
6
7
8
9
10

3.- Arrays.

3.1.- Arrays escalares.

Un Array es una colección de valores.
Los arrays pueden ser unidimensionales (vectores), bidimensionales (matrices o tablas) y multidimensionales (más de dos dimensiones).
Los arrays se utilizan ampliamente en el lenguaje PHP.
Se utiliza el delimitador [] para acceder a los diferentes elementos del vector.
Los arrays escalares se referencian por un subíndice de tipo número natural que va entre el delimitador, y que comienza en 0.
Podemos crearlo sin tener que declararlo previamente:

```
$dias[0]=31;
$dias[1]=28;
```

Una vez escrito eso, tenemos creado un vector de dos elementos, a los cuales accedemos por un subíndice que comienza a numerarse desde cero.

```
echo $dias[0];        //31
echo $dias[1];        //28
```

El vector, como podemos ver, puede ir creciendo en forma dinámica, es decir que si ahora hacemos:

```
$dias[2]=31;
```

el vector tiene 3 componentes.

También podemos ignorar el subíndice cuando asignamos los valores:

```
$dias[]=31;
$dias[]=28;
$dias[]=31;
```

Automáticamente comienza a numerarse desde cero.

Si necesitamos conocer el tamaño del vector en cualquier momento podemos llamar a la función count.

```
echo count($dias);   //3
```

Si queremos imprimir todos los elementos de un array podemos hacer:

```
<?php
$nombres[]="juan";
$nombres[]="pedro";
$nombres[]="ana";
for($f=0;$f<count($nombres);$f++)
{
 echo $nombres[$f];
 echo "<br>";
}
?>
```

La función sizeof(<nombre del vector>) es equivalente a count.

Otra forma de inicializar un vector es definirlo e inicializarlo simultáneamente:

```
$edades=array("menores","jovenes","adultos");
```

Estamos definiendo el vector edades con tres componentes, numeradas automáticamente de cero a dos.

Ejemplo de utilización de arrays escalares.

```
// Creamos un array escalar (basta con definir un elemento)
 $a[0]="Luisa";
 $a[2]="María";
// creamos un nuevo elemento de ese array
// esta vez sin tener en cuenta la posición consecutiva(número natural)
// si ponemos corchetes vacíos va añadiendo índices automaticamente
 $a[]="Julia";
// comprobamos que le ha asignado índice 3
 echo "El elemento ".$a[3]." tiene índice 3  <br>";
// ahora insertamos un nuevo elemento con indice 32
 $a[32]="Virginia";
// insertemos otro elemento de forma automática
 $a[]= "Teresa";
// la inserción se hará con indice 33
 print "Contenido del elemento de índice 33 ...".$a[33]."<br>";
// Intento imprimir el elemento 21 que no se ha definido
 print ("Elemento de índice 21 .... ". $a[21]. "<br>");
 print_r($a);
//Utilizamos una función predefinida que saca los elementos del array de una
      sola vez.
```

Salida:

```
El elemento Julia tiene índice 3
Contenido del elemento de índice 33 ...Teresa
Elemento de índice 21 ....
//Notice: Undefined offset: 21
            in C:\xampp\htdocs\PhpProject1\ArraysPHP.php on line 29
//Conclusión: aunque utilicemos saltos en los índice para asignar valores,
//las posiciones intermedias no están definidas.

Array ( [0] => Luisa [2] => María [3] => Julia [32] => Virginia [33] => Teresa )
```

3.2.- Arrays asociativos.

Este tipo de arrays no es muy común a otros lenguajes, pero en PHP son de uso indispensable en distintas situaciones. Un array asociativo permite acceder a un elemento del vector por medio de un subíndice de tipo string. Los elementos de un array asociativo pueden escribirse usando la siguiente sintaxis:

$a['indice']=valor;

En este caso estamos obligados a escribir el nombre del índice que habrá de ser una cadena y debe ponerse entre comillas.

Tanto en este supuesto como en el anterior, es posible –y bastante frecuente– utilizar como índice el contenido de una variable. El modo de hacerlo sería:

$a[$ind]=valor

En este caso, sea cual fuere el valor de la variable $ind, el nombre de la variable nunca se pone entre comillas.
Como ejemplo, consideremos que deseamos guardar en un vector el DNI, nombre y dirección de una persona. Empleando un vector con subíndice entero la solución sería:

```
<?php
$registro[]="11111111A”;
$registro[]="Martinez Camino, Carlos";
$registro[]="Cruz, 23";
?>
```

De esta forma debemos recordar que cuando deseamos mostrar el nombre de la persona debemos acceder al subíndice 1. Esto es sencillo si tenemos un vector con tres elementos, pero qué sucede si debemos almacenar 100 registros relacionados de 3 datos cada uno, en un vector?
Un vector asociativo se define de la siguiente forma:

```
<?php
$registro['dni']="11111111A";
$registro['nombre']="Martinez Camino, Carlos";
$registro['direccion']="Cruz, 23";
echo $registro['nombre'];
?>
```

Ahora vemos que para imprimir el nombre de la persona no debemos recordar una posición dentro de un vector sino un nombre de clave. Esto se hace indispensable cuando administramos un conjunto de datos grandes. En un vector asociativo cada componente está asociado a una clave.

Otras formas de crear un vector asociativo:

```
<?php
$registro=array('dni'=>'11111111A',
        'nombre'=>'Martinez Camino, Carlos',
 'direccion'=>'Cruz, 23’);
echo $registro['dni'];
?>
```

Ejemplo:

```
// Ejemplo con arrays asociativos.
// Creamos un array llamado $c con varios elementos
// Se trata de ejemplificar los datos de un país.
 $c['nombre']="Canadá";
 $c["continente"]="América";
 $c["superficie"]=9984670;
 $c["habitantes"]=36155487;
 $c["capital"]="Otawa";
//encadenamos variables para hacer una salida
 $salida="<H2> El país  ". $c["nombre"] .
       " está en el continente".$c["continente"]. "<br>";
 $salida .="Tiene una superficie de ".$c["superficie"]. " Km2.";
 $salida .=" y  ".$c["habitantes"]."  habitantes.<br>";
 $salida .=" Su capital es  ".$c["capital"];
 $salida .="</H2>";
 print $salida;
```

Salida:

```
El país Canadá está en el continente América
Tiene una superficie de 9984670 Km2. y 36155487 habitantes.
Su capital es Otawa
```

3.3.- Arrays bidimensionales.

Cada uno de los elementos se identifica por un nombre($nombre) seguido de dos ([]) que contienen los índices (en este caso son dos índices) del array.
Los índices pueden ser de tipo escalar -equivalen al número de fila y columna que la celda ocupa en la tabla– o puede ser asociativos lo que equivaldría en alguna medida a usar como índices los nombres de la fila y de la columna.
En este supuesto también se empiezan a numerar los arrays escalares a partir de CERO.

Arrays escalares
Los elementos de un array bidimensional escalar pueden escribirse usando una de estas sintaxis:
$a[][]=valor;

PHP asigna automáticamente como primer índice el valor que sigue al último asignado y, si es el primero que se define, le pondrá como índice (CERO).
Sea cual fuere el valor de primer índice al segundo se le asignará cero ya que es en este mismo momento cuando se habrá creado el primero y, por tanto, aún carecerá de elementos.

$a[xx][]=valor
asignamos un valor al primer índice(xx) y será el segundo quien se incremente en una unidad respecto al de valor más alto de todos aquellos cuyo primer índice coincide con el especificado.

$a[][xx]=valor
Ahora se modificaría automáticamente el primer índice y se escribiría el contenido (xx) como valor del segundo.

$a[xx][yy]=valor
se asignan libremente cada uno de los índices (xx e yy) poniéndoles valores numéricos.

Arrays asociativos didimensionales
Los elementos de un arraya asociativo bidimiensional se pueden escribir usando la siguiente sintaxis:
$a["indice1"]["indice2"]=valor
En este caso, los índices serán cadenas y se escribirán entre comillas.

```
$población[‘Alemania’][‘Berlin’]=4000000;
```

```
$población[‘Alemania’][‘Hamburgo’]=1763000;
$población[‘Francia’][‘Paris’]=7400000;
$población[‘Francia’][‘Lyon’]=850000;
$población[‘España’][‘Albacete’]=180000;
```

Arrays mixtos
PHP permite utilizar también arrays mixtos. Sería este el caso de que uno de ellos fuera escalar y el otro asociativo.
Igual que ocurría con los unidimensionales, también aquí podemos utilizar valores de variables como índices.

Ejemplo:

```
<?php
      /*Ejemplo de creación de un array bidimensional (matriz o tabla) con índices
      escalares.*/
       $pais[0][0] = "Méjico";
       $pais[0][1] = "Canadá";
       $pais[1][0] = "España";
       $pais[1][1] = "Francia";
       $pais[2][0] = "China";
       $pais[2][1] = "India";

   /* Otra forma de definirlo */
       $pais = array(
       array("Méjico", "Canadá"),
       array("España", "Francia"),
       array("China", "India")
       );

       /* y una mezcla de ambas */
       $pais[0] = array("Méjico", "Canadá");
       $pais[1] = array("España", "Francia");
       $pais[2] = array("China", "India");

       /* Para cualquiera de las definiciones */
       echo $pais[2][1]."<br>";
       echo $pais[0][0]."<br>";
?>
```

Salida:

```
India
Méjico
```

3.4.- Arrays multidimensionales.

PHP permite el uso de arrays con dimensión superior a dos. Para modificar la dimensión del arraya basta con ir añadiendo nuevos índices.
$a[x][y][z]=valor;
asignaría un valor al elemento de índices x, y, z de un arraya tridimensional y
$a[x][y][z][w]=valor;
haría lo mismo, ahora con un arraya de dimensión cuatro.

Pueden tener cualquier tipo de índices: escalares, asociativos y, también, mixtos.

La función **array()** multidimensional;
Para definir y asignar valores a una matriz puede usarse la función array(), que tiene la siguiente sintaxis:

```
$a= array (
índice  => valor,
..... ,
índice n => valor,
);
```

Por ejemplo:

```
$z=array (
        0 => 2,
        1 => "Carlos",
        2 => 34.7,
        3 => "17Alicia",
);
```

producirá igual resultado que:

```
$z[]=2;
$z[1]="Carlos";
$z[2]=34.7;
$z[3]= “17Alicia ";
```

La función array() permite escribir arrays de cualquier dimensión utilizando la técnica de anidado.
Si pretendemos escribir los elementos de este array:

```
$z[0][0]=34;
$z[0][1]=35;
$z[0][2]=36;
$z[1][0]=134;
$z[1][1]=135;
$z[1][2]=136;
```

podriamos hacerlo asi:

```
$z=array(
        0 => array (
                0 => 34,
                1 => 35,
                2 => 36,
        ),
        1 => array (
                 => 134,
                1 => 135,
                2 => 136,
        )
);
```

El anidado sucesivo permitiría generar arrays de cualquier dimensión.

Aunque en el ejemplo anterior nos hemos referido a un array escalar, idéntico procedimiento sería válido para arrays asociativos con solo cambiar los números por cadenas escritas entre comillas.

Este podría ser un ejemplo de arraya asociativo:

```
$z["a"]["A"]=34;
$z["a"]["B"]=35;
$z["a"]["C"]=36;
$z["b"]["A"]=134;
$z["b"]["B"]=135;
$z["b"]["C"]=136;
```

que podría definirse también de esta forma:

```
$z=array(
        "a" => array (
                "A" => 34,
                "B" => 35,
                "C" => 36,
                ),
        "b" => array (
                "A" => 134,
                "B" => 135,
                "C" => 136,
                )
);
```

3.5.- El bucle foreach.

El bucle foreach es específico de los arrays y aplicable a ellos tanto si son escalares como si son de tipo asociativo.
Tiene dos posibles opciones. En una de ellas lee únicamente los valores contenidos en cada elemento del array. En el otro caso lee además los índices del array.

Lectura de valores.
Utiliza la sintaxis:

```
foreach( $array as $var ){
        ...instrucciones...
}
```

,donde $array es el nombre del array(sin incluir índices ni corchetes), **as**es una palabra obligatoria y $var el nombre de una variable (puede ser creada al escribir la instrucción ya que no requiere estar previamente definida).
Las instrucciones escritas entre las {} permiten el tratamiento o visualización de los valores obtenidos.

Lectura de índices y valores.
Con una sintaxis como la que sigue se pueden leer no solo los valores de un array sino también sus índices.

```
foreach( $array as $indice =>$valor ) {
        ...instrucciones...
}
```

,donde **$array** es el nombre del array, **as** es una palabra obligatoria, **$indice** es el nombre de la variable que recogerán los índices, los caracteres => (son obligatorios) son el separador entre ambas variables y, por último, $**valor** es el nombre de la variable que recoge el valor de cada uno de los elementos del array.
Tanto esta función como la anterior realizan una lectura secuencial que comienza en el primer valor del array.

Arrays bidimensionales.
Cuando se trata de arrays bidimensionales la lectura de los valores que contienen sus elementos requiere el uso de dos bucles anidados.
Cuando un array de este tipo es sometido al bucle foreach se extrae como índice el primero de ellos y como valor un array unidimensional que tiene como índice el segundo del array original y como valor el de aquél.
La lectura de los valores de cada elemento requiere utilizar un segundo bucle que los vaya extrayendo a partir del array unidimensional obtenido por medio del bucle previo.

La sintaxis sería de este tipo:

```
foreach($array as $indice1=>$nuevoArray){
        foreach($nuevoArray as $indice2=>$valor){
                ..$indice1 es el primer índice...
                ..$indice2 es el segundo índice...
                .. $nuevoArray array extraído por el primer bucle
                ..$valor es el valor
                ....
                }
}
```

En el caso de arrays con dimensiones superiores sería necesario proceder del mismo modo, y habría que utilizar tantos bucles foreach como índices contuviera el array.
Un caso especial es el de los arrays de dos dimensiones asociativo, ya que el array extraído por el primer foreach(), al ser asociativo se pueden extraer los elementos del mismo por el nombre del índice, también ocurre accediendo por el índice en escalar. Lo que nunca estaría permitido es hacer un echo( ) del array completo.

Ejemplos:

```
<?php
        $a=array("a","b","c","d","e");

        foreach($a as $valor) {
           echo $valor,"<br>";
        };
```

```
        $b=array(
               "uno" =>"Primer valor",
              "dos" =>"Segundo valor",
              "tres" =>"Tercer valor",
        );

        foreach($b as $valor) {
           echo $valor,"<br>";
        };

        /*****************************/

        foreach($a as $indice=>$valor) {
           echo "Indice: ",$indice," Valor: ",$valor,"<br>";
};

        foreach($b as $indice=>$valor) {
              echo "Indice: ",$indice," Valor: ",$valor,"<br>";
};

        /*********************************/

        $z=array(
              0=> array (
                     0 => 34,
                     1 => 35,
                     2 => 36,
              ),
             1 => array (
                     0  => 134,
                     1 => 135,
                     2 => 136,
              )
        );

        /*
        foreach($z as $indice=>$valor) {
              echo "Indice: ",$indice," Valor: ",$valor,"<br>";
       };
        /*
         * Notice: Array to string conversion
           in C:\xampp\htdocs\PhpProjectBasico\BucleForEachPHP.php on line 56
         */
        foreach($z as $ind1=>$valor1) {
           foreach($valor1 as $ind2=>$valorReal) {
              echo "Ind. 1: ",$ind1,"  Ind. 2: ",$ind2,
                     "   Valor:",$valorReal,"<br>";
           };
        };

     ?>
```

Salida:

```
a
b
c
d
e
Primer valor
Segundo valor
Tercer valor
Indice: 0 Valor: a
Indice: 1 Valor: b
Indice: 2 Valor: c
Indice: 3 Valor: d
Indice: 4 Valor: e
Indice: uno Valor: Primer valor
Indice: dos Valor: Segundo valor
Indice: tres Valor: Tercer valor
Ind. 1: 0 Ind. 2: 0 Valor:34
Ind. 1: 0 Ind. 2: 1 Valor:35
Ind. 1: 0 Ind. 2: 2 Valor:36
Ind. 1: 1 Ind. 2: 0 Valor:134
Ind. 1: 1 Ind. 2: 1 Valor:135
Ind. 1: 1 Ind. 2: 2 Valor:136
```

3.6.- Funciones predefinidas en arrays.

array_push

array_push — Inserta uno o más elementos al final de un array existente.

int **array_push** (array *&$array* , mixed *$var* [, mixed *$*...])

Descripción

array_push() trata ***array*** como si fuera una pila y empuja la variable que se le pasa al final del ***array***. El tamaño del ***array*** será incrementado por el número de variables insertados. Tiene el mismo efecto que:

```
<?php
$array[] = $var;
?>
```

repetido por cada ***var***.

Nota: Si se utiliza **array_push()** para añadir un solo elemento en el array, es mejor utilizar ***$array[]*** = ya que de esta forma no existe la sobrecarga de llamar a una función.

Nota: **array_push()** generará un warning si el primer argumento no es un array. Esto difiere del comportamiento de ***$var[]*** donde un nuevo array será creado.

Parámetros

array

El array de entrada.

var

El valor a insertar.

Valores devueltos
Devuelve el nuevo número de elementos en un array.
Ejemplo #1 Ejemplo de array_push()

```
<?php
     $arrayPaises = array("Canadá", "Mongolia");
      array_push($arrayPaises, "Kenia", "Marruecos");
print_r($arrayPaises);
?>
```

El resultado del ejemplo sería:
Array ([0] => Canadá [1] => Mongolia [2] => Kenia [3] => Marruecos)

array_pop

array_pop — **Extrae** el último elemento del final del array

Descripción
mixed array_pop (array &$array)
array_pop() **extrae y devuelve** el último valor del array, acortando el array con un elemento menos. Si el array está vacío (o no es un array), se devolverá NULL. Además se producirá un Warning cuando se llame a un elemento que no es un array.

Nota: Esta función ejecutará un reset() en el puntero de array del array de entrada después de su uso.

Parámetros
array
El array de donde obtener el valor.
Valores devueltos
Devuelve el último valor del array. Si el array está vacío (o no es un array), se devolverá NULL.

Ejemplo de array_pop(), continuando con el anterior.

```
$pais = array_pop($arrayPaises);
      print ($pais."<br>");
print_r($arrayPaises);
```

Después de hacer esto, $arrayPaises solo tendrá 3 elementos:

```
Marruecos
Array ( [0] => Canadá [1] => Mongolia [2] => Kenia )
```

array_shift

array_shift — Quita un elemento del principio del array

Descripción
mixed **array_shift** (array *&$array*)
array_shift() , **quita** el primer valor del ***array*** y lo **devuelve**, acortando el ***array*** un elemento y corriendo el array hacia abajo. Todas la claves del array numéricas serán modificadas para que empiece contando desde cero mientras que los arrays con claves literales no serán modificados.

Nota: Esta función ejecutará un reset() en el puntero de array del array de entrada después de su uso.

Parámetros
array
El array de entrada.
Valores devueltos
Devuelve el valor quitado, o **NULL** si el ***array*** está vacío o no es un array.

Ejemplo de array_shift()

```
$pais = array_shif($arrayPaises);
print ($pais."<br>");
print_r($arrayPaises);
```

El resultado del ejemplo sería:

```
Canadá
Array ( [0] => Mongolia [1] => Kenia )
```

array_unshift

array_unshift — Añadir al inicio de un array uno a más elementos

Descripción
int **array_unshift** (array *&$array* , mixed *$value1* [, mixed *$...*])
array_unshift() añade los elementos pasados al inicio de ***array***. Observe que la lista de elementos se añade como un todo, por lo que los elementos añadidos permanecen en el mismo orden. Todas las claves numéricas del array serán modificadas empezando a contar desde cero mientras que las claves literales no se tocan.
Parámetros
array
El array de entrada.
value1
El primer valor a añadir al inicio.

Valores devueltos
Devuelve el nuevo número de elementos del ***array***.

Ejemplo #1 Ejemplo de array_unshift()

```
array_unshift($arrayPaises, "Argentina", "Nueva Zelanda");
    print_r($arrayPaises);
```

El resultado del ejemplo sería:

```
Array ( [0] => Argentina [1] => Nueva Zelanda [2] => Mongolia[3] => Kenia )
```

Funciones para posicionarse en el array.

Los arrays disponen de un cursor interno que es el encargado de posicionarse sobre cada uno de los elementos.

Este cursor podemos manejarlo con algunas funciones de PHP:

reset($nombrearray)	Coloca el cursor sobre el **primer** elemento del array.
next($nombrearray)	Mueve el cursor al **siguiente** elemento. Si no existen más elementos, la función devuelve false.
prev($nombrearray)	Mueve el cursor al elemento **anterior**. De no haberlo, la función devuelve false.
end($nombrearray)	Coloca el cursor sobre el **último** elemento del array.
current($nombrearray)	Devuelve el **valor** del elemento sobre el que está situado el cursor en ese momento. De no existir ningún valor, devolverá false.
key($nombrearray)	Devuelve el **índice** del elemento sobre el que está situado el cursor en ese momento. Si no existe ningún elemento, devolverá NULL.

Eliminar elementos de un array.

Si necesitamos eliminar alguno de los elementos almacenados en un array **por su posición**, podemos usar la función **unset().**

Esta función espera que le pasemos como parámetro el nombre del array y el índice del elemento que deseamos borrar, quedando la expresión de la siguiente forma:

unset($nombreArray[índice]);
Ejemplo:

```
unset($arrayPaises[1]);
print_r($arrayPaises);
```

Salida:

```
Array ( [0] => Argentina [2] => Mongolia [3] => Kenia )
```

unset() reordena los índices tras borrar uno de los elementos del array.
O sea, que si eliminamos el elemento nº 2 del array, entonces el tercer elemento ocupará su lugar, el cuarto pasará al tercero y así sucesivamente.
unset($nombrearray);
Esta función borra toda la estructura del array.

```
unset($arrayPaises);
print_r($arrayPaises);
```

Salida:

```
Notice: Undefined variable: arrayPaises in
C:\xampp\htdocs\PhpProjectBasico\FuncionesPredefinidasArrays.php on line xx
```

Si lo que queremos es eliminar únicamente los elementos usaríamos el constructor array() sin pasarle ningún parámetro.
$nombrearray = array();

Funciones para ordenar arrays.

Funciones disponibles para ordenar arrays.

sort()

Es la función más básica para ordenar arrays en PHP. Ordena los valores del array de menor a mayor.
Ejemplo:

```
//Ordenar de menor a mayor
 $paises = array("España", "Francia", "Canadá", "Méjico", "Venezuela", "India");
       sort($paises);
         foreach ($paises as $key => $val) {
       echo "país[" . $key . "] = " . $val . "<br>";
        }
```

Salida:

```
país[0] = Canadá
país[1] = España
país[2] = Francia
país[3] = India
país[4] = Méjico
país[5] = Venezuela
```

rsort()

Esta función ordena el array por valores. La "r" delante quiere decir que ordena en orden reverso, de mayor a menor.

```
//Ordenar de mayor a menor (Reverse order)
rsort($paises);
      foreach ($paises as $key => $val) {
echo "país[" . $key . "] = " . $val . "<br>";
      }
```

Que daría como respuesta:

```
país[0] = Venezuela
país[1] = Méjico
país[2] = India
país[3] = Francia
país[4] = España
país[5] = Canadá
```

ksort()

También podemos ordenar un array por el índice o llave, que quiere decir que en lugar de ordenar atendiendo a los valores, se ordenaría atendiendo al índice que tienen. No tiene mucho sentido aplicarlo a arrays escalares cuyo índice es un número. Ejemplificaremos con arrays asociativos.

En el array siguiente vemos que tenemos índices como "x", "e", "t", en lugar de números. Lo que hará este tipo de ordenación es fijarse en esos índices para poner el array ordenado por ellos.

```
//Ordenar arrays por su índice
$paises2 = array("i"=>"Namibia", "x"=>"Mauritania", "e"=>"Honduras",
"s"=>"Suecia", "t"=>"Polonia");
ksort($paises2);
```

```
foreach ($paises2 as $key => $val) {
        echo "país[" . $key . "] = " . $val . "<br>";
 }
```

Esto dará como resultado esta ordenación:

```
país[e] = Honduras
país[i] = Namibia
país[s] = Suecia
país[t] = Polonia
país[x] = Mauritania
```

krsort()

También podemos ordenar por índices pero en sentido inverso. Es decir, por índices pero de mayor a menor.

```
//ordenar por índice o clave, pero en orden inverso
krsort($paises2);
      foreach ($paises2 as $key => $val) {
echo "país[" . $key . "] = " . $val . "<br>";
      }
```

En este caso el resultado sería el siguiente:

```
país[x] = Mauritania
país[t] = Polonia
país[s] = Suecia
país[i] = Namibia
país[e] = Honduras
```

asort()

Un problema derivado de ordenar arrays asociativos por su valor es que no 'respeta' el índice asociado, transformándolo en un escalar.

Ejemplo:

```
$paises2 = array("i"=>"Namibia", "x"=>"Mauritania", "e"=>"Honduras",
"s"=>"Suecia", "t"=>"Polonia");
sort($paises2);
      foreach ($paises2 as $key => $val) {
echo "país[" . $key . "] = " . $val . "<br>";
      }
```

Salida:

```
país[0] = Honduras
país[1] = Mauritania
país[2] = Namibia
país[3] = Polonia
país[4] = Suecia
```

Para solventar este problema utilizamos esta función que ordena los elementos de un array, pero manteniendo la correlación entre índices y valores a los que están asociados. Ordena por valores. Se utiliza generalmente en arrays asociativos.

```
//ordenar arrays asociativos manteniendo los índices
//asort()
$capitales = array("España" => "Madrid", "Mauritania" => "Nuakchot",
"India" => "Nueva Delhi", "Paraguay" => "Asunción", "Hungría"=> "Budapest");
asort($capitales);
 foreach ($capitales as $key => $val) {
        echo "País: ".$key ." -> Capital: ". $val . "<br>";
       }
```

Dará como resultado el siguiente orden de array:

```
País: Paraguay -> Capital: Asunción
País: Hungría -> Capital: Budapest
País: España -> Capital: Madrid
País: Mauritania -> Capital: Nuakchot
País: India -> Capital: Nueva Delhi
```

arsort()

Es lo mismo que asort(), pero realiza el orden en inverso de los valores de los arrays.

Ahora ejemplificaremos con escalares para ver cómo los índices no se reordenan, sino que se mantienen asociados a su valor.

```
//ordenar arrays escalares manteniendo los índices numéricos
//arsort()
$capitales2 = array("Madrid","Nuakchot","Nueva Delhi","Asunción","Budapest");
arsort($capitales2);
 foreach ($capitales2 as $key => $val) {
        echo $key ." -> ". $val . "<br>";
  }
```

El resultado obtenido es este:

```
2 -> Nueva Delhi
1 -> Nuakchot
0 -> Madrid
4 -> Budapest
3 -> Asunción
```

natsort()

Esta función hace una ordenación natural de los elementos del array, es decir, ordena tal como lo haría una persona. La función natsort mantiene la asociación clave – valor que no hacía sort(), pero tiene un comportamiento idéntico a asort().

```
//ordenar arrays escalares manteniendo el orden natural con el índice original
//natsort()
$capitales2 = array("Madrid","Nuakchot","Nueva Delhi","Asunción","Budapest");
natsort($capitales2);
foreach ($capitales2 as $key => $val) {
        echo $key ." -> ". $val . "<br>";
 }
```

Esto daría como respuesta:

```
3 -> Asunción
4 -> Budapest
0 -> Madrid
1 -> Nuakchot
2 -> Nueva Delhi
```

array_multisort()

Se suele utilizar en arrays multidimensionales o cuando queramos ordenar por más de un valor asociativo.

Ejemplo:

```
<?php
$Paises[] =
array(“Nombre”=>”España”,”Capital”=>”Madrid”,"Poblacion"=>46570000);
$Paises[] =
array(“Nombre”=>”Francia”,”Capital”=>”París”,"Poblacion"=>67120000);
$Paises[] =
array(“Nombre”=>”Canadá”,”Capital”=>”Otawa”,"Poblacion"=>36710000);

foreach ($Paises as $key => $Pais) {
        $Capital[$key]  = $Pais[“Capital”]; // columna de capitales
        $Poblacion[$key] = $Pais[“Poblacion”]; //columna de población
    }
 //ordenamos ascendente por la columna que queramos
array_multisort($Capital, SORT_ASC, $Paises);
print_r($Paises);
?>
```

En el ejemplo anterior tenemos el array $datos con la información que queremos ordenar; la idea es poner las columnas que interesan ordenar en **arrays independientes** de esta forma puedes ordenar por la columna que quieras o por varias a la vez.
Luego usamos la función **array_multisort** y le pasamos la columna por la que queremos **ordenar**, si es ascendente o descendente y finalmente el array original donde se guardara el array ordenado.

Ejercicios propuestos de arrays.

Ejercicios de arrays escalares.

1.- De los N elementos de un array con números de una dimensión, calcular su suma, máximo, mínimo y media aritmética.

Ejemplo:

2	4	5	1	-4	6	5	3	1	9

Solución:
Suma: 32.
Máximo: 9.
Mínimo: -4.
Media: 3,2.

2.- Disponemos de 2 arrays con números de una dimensión ordenados por su contenido de menor a mayor y queremos obtener otro array de una dimensión con los elementos de los anteriores y también ordenado.

Ejemplo:

2	4	5	7	8	12	15	23	34	49

1	4	6	8	11	25	27	34	51	49

Solución:

1	2	4	4	5	6	7	8	8	11	12	15	23	25	27	34	34	49	49	51

3.- Hacer un script en PHP que reconozca si una frase es palíndrome (capicúa al eliminar los espacios).
Existen muchas frases que lo cumplen:

DABALE ARROZ A LA ZORRA EL ABAD
NO DESEO YO ESE DON
ANA LLEVA AL OSO LA AVELLANA

Ejercicios arrays didimensionales.

4.- Crear un array de una dimensión con la suma de los valores de las columnas de una matriz de N x N (array de dos dimensiones) cuyo contenido son números enteros.

Ejemplo:

2	2	1	1	7
0	1	5	2	1
5	2	6	4	2
1	5	3	6	3
1	6	3	0	0

Solución:

9	16	18	13	13

5.- Sumar 2 matrices con números enteros de N x N elementos, dando como resultado otra de N X N.
Ejemplo:

2	2	1	1	7
0	1	5	2	1
5	2	6	4	2
1	5	3	6	3
1	6	3	0	0

+

5	0	2	3	1
0	9	4	1	7
1	3	8	2	8
2	5	0	4	2
7	6	0	1	1

=

7	2	3	4	8
0	10	9	3	8
6	5	14	6	10
3	10	3	10	5
8	12	3	1	1

Para sumar dos matrices, se suman los elementos por su posición correspondiente.

6.- Una matriz se dice "casi vacía" cuando tiene la mayoría de sus elementos nulos (no cero), para evitar el desperdicio de memoria de tal estructura se suele almacenar en un array de una dimensión los elementos no nulos.
Ejemplo:

2		1		
		5		1
5			4	
		3		

En tal caso determinar:

a) Estructura del array para cumplir dicho cometido.

b) Script en PHP de búsqueda de un elemento cualquiera, obteniendo la información de la fila y columna que ocuparía en la matriz o la no presencia del elemento en la misma.

Ejercicios propuestos arrays asociativos.

7.- Define un array asociativo en el que los índices sean el nombre de un país de la UE, y su contenido el nombre del presidente del gobierno del mismo. Ejemplifica 7 países. Después recorre dicho array y saca los nombres de los presidentes. A continuación realiza el mismo listado pero obteniendo la asociación del nombre del país y el presidente.

8.- Crea un array asociativo que describa información relativa a cinco países. El índice será el nombre del país.
La información registrada: continente en el que se encuentra, población, superficie en km^2 e idioma oficial.
Se pide:

- Hacer un listado de todos los países con toda la información.
- Sacar la información anterior ordenada por nombre de país.
- Obtener el país con más habitantes.
- Obtener el país más pequeño en superficie.
- Insertar un nuevo país.
- Quitar el que ocupe el tercer puesto.
- Volver a hacer el listado para comprobar que los cambios han surtido efecto.
- Sacar un listado completo de los países ordenado por número de habitantes de mayor a menor.

4.- Funciones de usuario.

Ejemplos de utilización de funciones.
Ejemplo 1:

```
<?php
$a=5; $b=47;
function a1(){
	echo 'Este es el valor de $a en la función a1: ',$a,'<br>';
	echo 'Este es el valor de $b en la función a1: ',$b,'<br>';
}
a1();
function a2(){
	global $a;
	echo 'Este es el valor de $a en la función a2: ',$a,'<br>';
	echo 'Este es el valor de $b en la función a2: ',$b,'<br>';
}

a2();

function a3(){
	global $a;
	$a +=45;
	$b -=348;
	echo 'Este es nuevo valor de $a en la función a3: ',$a,'<br>';
	echo 'Este es el valor de $b en la función a3: ',$b,'<br>';
}

a3();
echo 'El valor de de $a HA CAMBIADO después de ejecutar a3 es: ',$a,'<br>';
echo 'El valor de de $b NO HA CAMBIADO después de ejecutar a3 es:
',$b,'<br>';
function a4(){
	print 'La superglobales sí están:
	'.$_SERVER['SERVER_NAME'].'<br>';
}
a4();
?>
```

Salida:

```
Este es el valor de $a en la función a1:
Notice: Undefined variable: a
in C:\xampp\htdocs\PhpProject1\ejemploFuncionesUsuario.php on line 16

Este es el valor de $b en la función a1:
Notice: Undefined variable: b
in C:\xampp\htdocs\PhpProject1\ejemploFuncionesUsuario.php on line 17

Este es el valor de $a en la función a2: 5
Este es el valor de $b en la función a2:
Notice: Undefined variable: b
in C:\xampp\htdocs\PhpProject1\ejemploFuncionesUsuario.php on line 23

Notice: Undefined variable: b
in C:\xampp\htdocs\PhpProject1\ejemploFuncionesUsuario.php on line 31
Este es nuevo valor de $a en la función a3: 50
Este es el valor de $b en la función a3: -348
```

```
//Aunque dio error en la asignación, evalúa $b a 0 y puede hacer el echo de la misma.
El valor de de $a HA CAMBIADO después de ejecutar a3 es: 50
El valor de de $b NO HA CAMBIADO después de ejecutar a3 es: 47
La superglobales sí están: localhost
```

Ejemplo 2.-

```
<?php
$a=-13; $b=7482; $c="Carlos";
function a1($a=56, $b=25){
        echo 'El valor  de $a en la función a1: ', $a,'<br>';
        echo 'El valor  de $b en la función a1: ', $b,'<br>';
}
a1();
echo 'El valor de $a después de ejecutar la función es: ',$a,'<br><br>';

function fun1($x,$y,$z){
        print "Valor de la variable x: ".$x."<br>";
        print "Valor de la variable y: ".$y."<br>";
        print "Valor de la variable z: ".$z."<br>";
}

fun1(14,"Elena",23.4);
fun1(49.3,"Carlos",78,"Gloria",456);
fun1("Ana","Lino");
@fun1("Nuevo Ana","Nuevo Lino");
        @PHP error control operator: the at sign (@). When prepended to an expression
        in PHP, any error messages that might be generated by that expression will be
        ignored.
fun1("La luna","","nueva");
fun1($a,$b,$c);
?>
```

Salida:

```
El valor de $a en la función a1: 56
El valor de $b en la función a1: 25
El valor de $a después de ejecutar la función es: -13

Valor de la variable x: 14
Valor de la variable y: Elena
Valor de la variable z: 23.4
Valor de la variable x: 49.3
Valor de la variable y: Carlos
Valor de la variable z: 78

Fatal error: Uncaught ArgumentCountError: Too few arguments to function fun1(), 2 passed in
C:\xampp\htdocs\PhpProject1\ejemploFuncioneUsuario2.php on line 30 and exactly 3 expected
in C:\xampp\htdocs\PhpProject1\ejemploFuncioneUsuario2.php:22 Stack trace: #0
C:\xampp\htdocs\PhpProject1\ejemploFuncioneUsuario2.php(30): fun1('Ana', 'Lino') #1 {main}
thrown in C:\xampp\htdocs\PhpProject1\ejemploFuncioneUsuario2.php on line 22
```

Ejemplo 3.-

```
<? php
$a=3; $b=2;

function a1(&$a,$b){
        $a=pow($a,2);
        $b=pow($b,3);
        echo "El cuadrado de a dentro de la función es: ",$a, "<br>";
        echo "El cubo de b dentro de la función es: ",$b, "<br><br>";
}

a1($a,$b);
echo "Al salir de la función a conserva la modificación: ",$a, "<br>";
echo "Por el contrario, b no la conserva: ",$b, "<br><br>";

?>
```

Salida:

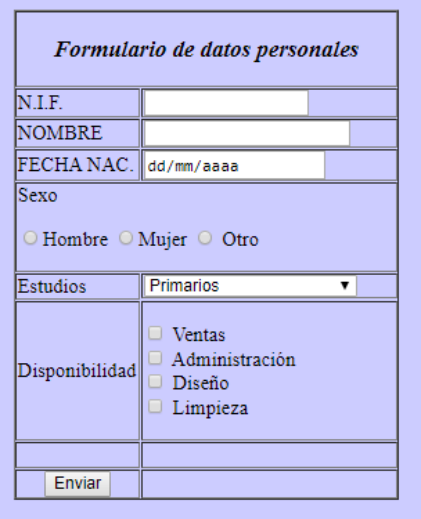
```
El cuadrado de a dentro de la función es: 9
El cubo de b dentro de la función es: 8

Al salir de la función a conserva la modificación: 9
Por el contrario, b no la conserva: 2
```

5.- Formularios.

En la arquitectura cliente-servidor la forma más usual de intercambiar información a través de una red es hacerlo a través del navegador del equipo cliente conectando a un servidor.
El equipo cliente solicita una página de inicio de un servidor, adjunta datos y manda nuevamente esa página (formulario) al servidor para su procesamiento.
La página de inicio tiene formato html.
En el servidor habrá un script de php que analizará los datos enviados.

Cuando un usuario pulsa el botón enviar de un formulario, la información que contenían sus campos es enviada a una dirección URL desde donde tendremos que recuperarla para tratarla de alguna manera. Por ejemplo, si realiza una compra, tendremos que recuperar los datos para completar el proceso de pago. La información del formulario "viaja" almacenada en variables que podremos recuperar y utilizar mediante PHP. Una de las formas de recuperación consiste en usar **$_REQUEST**.

Ejemplo de aplicación:
1.- Realizar en HTML el formulario que se muestra:



Dentro de los Estudios:

2.- Seguir las instrucciones del profesor para que forme parte de un fichero en un determinado proyecto.

3.- Crear un script de PHP que recoja los datos del formulario y los envíe de nuevo al cliente.

4.- Hacer un segundo script en el que el envío de la información se haga mediante formato de tabla.

[PhpProjectPruebaFormulario]

6.- Cookies.

PHP soporta cookies HTTP de forma transparente. Las Cookies son un mecanismo por el cual se almacenan datos en el browser remoto y así rastrear o identificar a usuarios que vuelven. Se pueden configurar Cookies usando la función setcookie() o setrawcookie(). Las Cookies son parte del header HTTP, así es que setcookie() será llamada antes que cualquier otra salida sea enviada al browser. Esta es la misma limitación que tiene la función header().

bool setcookie (string $name [, string $value [, int $expire =]])

Todos los argumentos exceptuando el argumento name son opcionales. También puede reemplazar un argumento con un string vacío ("") para saltárselo. Ya que el argumento expire es un entero, no puede pasarse por alto con un string vacío, en su lugar utiliza un cero ( ).

name
El nombre de la cookie.

value
El valor de la cookie. Este valor se guarda en el ordenador del cliente. Asumiendo que el name es 'cookiename', este valor se obtiene con $_COOKIE['cookiename'].

expire
El tiempo en el que expira la cookie. Es una fecha Unix por tanto está en número de segundos a partir de 1 de enero de 1970 (fecha de lanzamiento del sistema operativo). Lo más habitual es utilizar la función time() más el número de **segundos** que queremos que dure la cookie. También se puede utilizar la función mktime().
Ejemplo_
time()+60 *60*24*3 configurará la cookie para expirar en 3 días. Si se pone 0 o se omite, la cookie expirará al final de la sesión (al cerrarse el navegador).

setcookie() define una cookie para ser enviada junto con el resto de las cabeceras de HTTP. Al igual que otras cabeceras, las cookies deben ser enviadas antes de que el script genere ninguna salida (es una restricción del protocolo). Esto implica que las llamadas a esta función se coloquen antes de que se genere cualquier salida.

Una vez que han sido enviadas las cookies, se puede acceder a ellas en la próxima carga de la página gracias a los arrays $_COOKIE o $HTTP_COOKIE_VARS. Las superglobales tales como $_COOKIE están disponibles a partir de PHP 4.1. . El valor de las cookies también está en $_REQUEST.
Debemos asegurarnos dónde guarda nuestro navegador las cookies que se instalan en la máquina. En Chrome, nuestras cookies las guarda bajo el epígrafe de ‘localhost’.

Envío de una cookie:

```
<?php
        $value = 'Una Cookie';
        setcookie("TestCookie", $value);
        setcookie("TestCookie", $value, time()+3600 );  //expira en una hora
?>
```

Leer el contenido de una cookie:

```
<?php
// Imprimir una cookie individual
        echo $_COOKIE["TestCookie"];

//Otra manera de depurar/probar es viendo todas las cookies
        print_r($_COOKIE);
?>
```

Borrado de cookies:

```
<?php
//establecer la fecha de expiración a una hora atrás
        setcookie ("TestCookie", "", time() - 3600  );
?>
```

Ejemplo:

Hacer Cookie.

```
<?php
      //HACER COOKIE
      # setcookie genera una cookie en el ordenador del cliente
setcookie("cookie1","Mi Cookie",time()+3600);

#cookie definida como Array
       $valores=array("Asia","India","Nueva Delhi","1324000000",”3287263”);
      # Introducimos los datos en un array asociativo
      # los índices no van entrecomillados
       setcookie("cookie3[continente]",$valores[0],time()+3600 );
setcookie("cookie3[pais]",$valores[1],time()+3600  );
       setcookie("cookie3[capital]",$valores[2],time()+3600  );
setcookie("cookie3[habitantes]",$valores[3],time()+3600  );
       setcookie("cookie3[superficie]",$valores[4],time()+3600  );
      /* la variable superglobal $_COOKIE['cookie3'] contiene un array, por ello
      la lectura de sus valores debe hacerse después considerando que se trata de
     un array bidimensional */
?>
```

Leer Cookie.

```
<?php
      // LEER COOKIE
      //Comprobar si ha llegado alguna Cookie
       print_r($_COOKIE);
      // Después se quita.

      //Debemos saber el nombre de las Cookies para extraerlas.
       echo "Esta es la cookie:",$_COOKIE['cookie1'],"<br>";

# Obtener las cookies del array
       if (isset($_COOKIE['cookie3']) ) {
```

```
        foreach($_COOKIE['cookie3'] as $indice=>$valor){
echo "$indice ==> $valor","<br>";
        }
    }
?>
```

Ejemplo “contador de visitas”.

```
<?php
    $numero=0;
    if(isset ($_COOKIE['visitante']) )
      $numero=$_COOKIE['visitante'];
    $numero+=1;
    setcookie("visitante",$numero,time()+8640);
    if($numero==1){print "Hola nuevo visitante";}
    if($numero>1){print "Es la $numero ª vez que visitas esta página";}
?>
```

Surge la duda de si una cookie sin tiempo de expiración dará siempre el mensaje de nuevo visitante o la sesión a la que pertenece le dará persistencia.
La conclusión es que la cookie se inhabilita al cerrar el navegador y no nos recuerda. Mientras el navegador esté abierto, la cookie existe siempre.

Idea asociada: buscar la cookie en el sistema.

7.- Sesiones.

Las sesiones, en aplicaciones web realizadas con PHP y en el desarrollo de páginas web en general, nos sirven para almacenar información que se guardará durante toda la visita de un usuario a una página web.
Las sesiones son una manera de guardar información específica para cada usuario durante toda su visita. Cada usuario que entra en un sitio abre una sesión, que es independiente de la sesión de otros usuarios. En la sesión de un usuario podemos almacenar todo tipo de datos, como su nombre, productos de un hipotético carrito de la compra, preferencias de visualización o trabajo, páginas por las que ha pasado, etc. Todas estas informaciones se guardan en lo que denominamos variables de sesión.
PHP dispone de un método bastante cómodo de guardar datos en variables de sesión, y de un juego de funciones para el trabajo con sesiones y variables de sesión.
Para cada usuario del sitio web, PHP internamente genera un identificador de sesión único, que sirve para saber las variables de sesión que pertenecen a cada usuario.
Si el servidor quisiera conservar los datos de la sesión de un usuario, deberá guardar en una cookie (llamada específicamente cookie de sesión), el identificador de esa sesión. Posteriormente, esa cookie viajará hasta el navegador del cliente para ser recordado en una próxima visita.
Normalmente, los datos obtenidos en la sesión se guardan en el servidor.
Cuando un usuario desde un navegador accede a un sitio web, el script de inicio abre una sesión, comprueba si lleva una cookie de sesión asociada, en tal caso extrae el código de sesión y busca ese código entre las sesiones guardadas, si lo encuentra, la abre; si no lo encuentra es que la sesión expiró o se trata de la primera visita.
Dentro de **php.ini** existe una directiva ***session.cookie-lifetime*** que especifica la duración de la cookie de sesión. Si está a 0 (segundos), al cerrar el navegador la cookie expirará y, por tanto, la sesión. Si deseamos que la invalidación de la sesión la controle el programador deberá fijarse a un valor muy alto (incluso días) dependiendo de las características del web site.

7.1.- Trabajo con sesiones en PHP

Cuando queremos utilizar variables de sesión en una página tenemos que iniciar la sesión con la siguiente función:
session_start ()

Inicia una sesión para el usuario o continúa la sesión que pudiera tener abierta en otras páginas. Al hacer session_start(), PHP internamente recibe el identificador de sesión almacenado en la cookie o el que se envíe a través de la URL. Si no existe tal identificador de sesión, simplemente lo crea.
Nota: Si en el **php.ini** se ha definido la variable **session.auto_start = 1** se inicializa automáticamente la sesión en cada página que visita un usuario, sin que se tenga que hacer el session_start().

Una vez inicializada la sesión con session_start() podemos a partir de ahora utilizar variables de sesión, es decir, almacenar datos para ese usuario, que se conserven durante toda su visita o recuperar datos almacenados en páginas que haya podido visitar.
La sesión se tiene que inicializar antes de escribir cualquier texto en la página. Esto es importante y de no hacerlo así corremos el riesgo de recibir un error, porque al iniciar la sesión se deben leer las cookies del usuario, algo que no se puede hacer si ya se han enviado las cabeceras del HTTP.
Pero los *include* no cumplen con esta restricción, es decir, si debemos incluir un fichero en nuestro script de PHP debe hacerse antes de abrir la sesión.
Una vez iniciada la sesión podemos utilizar variables de sesión a través de **$_SESSION**, que es un array **global asociativo**, donde se accede a cada variable a partir de su nombre, de este modo:
$_SESSION["nombre_de_variable"]

Importante!! Si nuestro script requiere incluir algún fichero, la orden include debe ir antes del session_start();
Nota : En versiones actuales ya no es necesario?

Ejemplo de código para definir una variable de sesión:

```
<?php
session_start();
?>
<html>
<head>
<title>Generar variable de sesión</title>
</head>
<body>
<?php
$_SESSION["mivariabledesesion"] = "Hola este es el valor de la variable de
sesión";
?>
</body>
</html>
```

Como se puede ver, es importante inicializar la sesión antes de hacer otra cosa en la página. Luego podremos definir variables de sesión en cualquier lugar del código PHP de la página.
Para leer una variable de sesión se hace a través del mismo array asociativo $_SESSION. Es tan sencillo como haríamos para utilizar cualquier otra variable, lo único es que tenemos que haber inicializado la sesión previamente. Y por supuesto, que la variable que deseamos acceder exista previamente.

```
<?php
session_start();
?>
<html>
<head>
<title>Leo variable se sesión</title>
```

```
</head>
<body>
Muestro esa variable:
<?php
echo $_SESSION["mivariabledesesion"];
?>
</body>
</html>
```

Como se puede ver, al inicio del código hemos inicializado la sesión y luego en cualquier parte del código podríamos acceder a las variables de sesión que tuviésemos creadas.

Nota: si intentamos acceder a una variable de sesión con $_SESSION que no ha sido creada obtendremos otro mensaje de error:
Notice: Undefined index: mivariabledesesion,
que es el mismo que si intentamos acceder a cualquier elemento de un array que no existe.

Otras funciones útiles para la gestión de sesiones son:

session_id()
Nos devuelve el identificador de la sesión

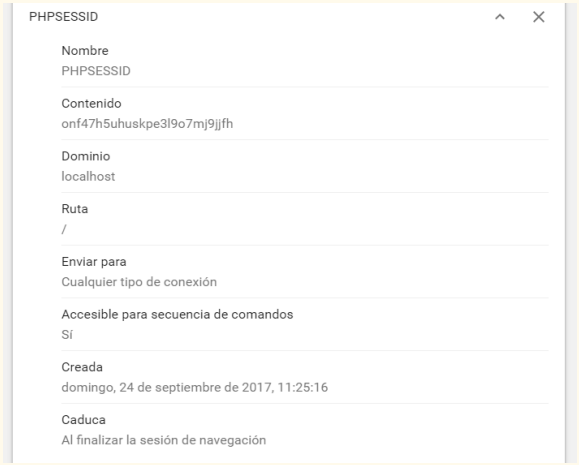
session_destroy()
Da por abandonada la sesión eliminando todas las variables e identificador asociados al usuario. No funciona. No usar

unset('variable') / unset()
Abandona una variable sesión concreta.
También tiene la opción de borrar la sesión completa.

Ejercicio propuesto.-
Implementar un contador de accesos con sesiones en PHP.

```
<?php
        session_start();
if(!isset($_SESSION["visitcounter"])){
        echo "Es tu primera visita <br>";
                $_SESSION["visitcounter"]=1;
      }
      else{
                $contador_visitas=$_SESSION["visitcounter"]+1;
        echo "Es la ",$contador_visitas,"ª vez que accedes <br>";
                $_SESSION["visitcounter"]=$contador_visitas;
                /* Escribimos el identificador de sesión */
                echo "Identificador de sesión: ", session_id(),"<br>";
        }
     ?>
```

Nos preguntamos : ¿Cuándo desaparece una sesión?
Al investigar la cookie asociada:



Se supone que debería caducar al cerrar el navegador, pero observamos que existen configuraciones del navegador en las que no es así.

Buscando la configuración en el ***php.ini*** está definida como :
; Lifetime in seconds of cookie or, if 0, until browser is restarted.
; http://php.net/session.cookie-lifetime
session.cookie_lifetime=0

Por tanto, debería desaparecer al cerrar el navegador, si no es así.
Buscar explicación.
La cookie persiste.

Para destruir la sesión completamente, como por ejemplo para desconectar al usuario, el id de sesión también debe ser destruido, por tanto la cookie de sesión se debe borrar, setcookie() se puede usar para eso poniéndole un valor negativo, pero utilizando su versión completa.

[Solución aportada en el phpProjectSession]

7.2.- Transferencia de datos entre páginas PHP.

Una forma más de paso de variables de una página a otra son las sesiones. Una sesión es una estructura de datos que se crea en el servidor y puede usarse sin que el usuario de la Web tenga conocimiento alguno de ello.
Pero ¿en que se diferencian las sesiones de los métodos POST y GET? Los métodos POST y GET permiten que los usuarios asignen valores a variables, y también permiten que la propia página tome valores internos, por ejemplo de una base de datos, y opere con ellos. Los valores emitidos por POST y GET pueden tener un origen conocido para el usuario y también su destino puede ser conocido. De cualquier forma, el usuario podrá tener conocimiento de las variables que se envían y de sus valores. Pues bien, con las sesiones podemos hacer lo mismo, pero con una diferencia, la variable de sesión será recuperable en cualquier parte del sitio Web sin tener que crear enlaces de pasos de variable o formularios con métodos GET o POST, por lo que el usuario no sabrá ni cuando se crea (aunque lo pueda intuir) ni dónde o cuándo se recupera y usa la variable. Simplemente cuando pasas de una página a otra mediante un enlace normal y corriente, tendrás la variable disponible para usarla. Sin las variables de sesión no serian posible los sistemas seguros de identificación.
Como normativa de los últimos años, al usuario de una web que usa cookies se le debe informa de tal eventualidad. Podremos ver las características de

dicha cookie en la configuración de nuestro navegador pero no el contenido de la sesión que siempre permanecerá en el servidor.

Ejemplo:

```
----- sesion1.php
<?php
session_start();
?>
<html>
<head>
<title>Generar variable de sesión</title>
</head>
<body>
<form method="POST" action="sesion1.php">
<p>Valor para la variable de sesión <input type="text" name="valorsesion"
size="20"></p>
<p><input type="submit" value="Creamos variable de sesion" name="B1"></p>
</form>

<?php
$valorsesion = $_POST['valorsesion'];
$_SESSION["var_sesion"] = $valorsesion;
?>
<p><a href = "sesion2.php">Ver valor de sesion</a></p>
</body>
</html>

----- sesion2.php
<?php
session_start();
?>

<html>
<head>
<title>Leo variable se sesión</title>
</head>
<body>
Muestro el valor que toma la variable sesión: <br>
<?php
$valor_sesion = $_SESSION["var_sesion"];
echo $valor_sesion;
?>

<p><a href = "sesion3.php">Ver la variable en otra pagina</a></p>
</body>
</html>

----- sesion3.php
<?php
session_start();
?>

<html>
```

```
<head>
<title>Leo variable se sesión en otra pagina</title>
</head>
<body>

Muestro el valor que toma una variable cualquiera junto con el
de la variable sesión: <br>
<?php
$unacualquiera = "lo que sea";
$valor_sesion = $_SESSION["var_sesion"];
echo"Vemos una variable de esta pagina <b>$unacualquiera</b>
y la variable de sesion <b>$valor_sesion</b>";
?>
<p><a href = "sesion4.php">Eliminar variable de sesion</a></p>
</body>
</html>
```

----- sesion4.php

```
<?php
session_start();
session_destroy();
?>

<html>
<head>
<title>Variable de sesion eliminada.</title>
</head>
<body>
Intento escribir la variable de sesion, pero ya no aparece: <br>
<?php
$valor_sesion = $_SESSION["var_sesion"];
echo $valor_sesion;
?>

<p><a href = "sesion1.php">Vuelvo a crear variable de sesion.</a></p>
</body>
</html>
```

8.- Inclusión de ficheros.

La sentencia **include** incluye y evalúa el archivo especificado en el punto exacto del fichero php. Si contiene variables, sólo serán conocidas a partir de este punto.

Los archivos son incluidos con base en la ruta de acceso dada o, si ninguna es dada, el include_path especificado. Si el archivo no se encuentra en el include_path, include finalmente verificará en el propio directorio del script que hace la llamada y en el directorio de trabajo actual, antes de dar error.
El operador include emitirá una advertencia si no puede encontrar un archivo, éste es un comportamiento diferente al de require, el cual emitirá un error fatal.

require_once

La sentencia require_once es idéntica a require excepto que PHP verificará si el archivoya ha sido incluido y si es así, no se incluye (require) de nuevo.
También existe la versión include_once.

Ejemplos:

#vars.php
```
<?php
	$color = 'verde';
	$fruta = 'manzana';
?>
```

#test.php
```
<?php
	echo "Una $fruta $color <br>";
	include 'vars.php';
	echo "Una $fruta $color";
?>
```

#test2.php
```
<?php
function foo()
{
global $color;//El valor de la variable la expande fuera de la función
include_once 'vars.php';
echo "Una $fruta $color <br>";
}

foo();
 echo "Una $fruta $color";
?>
```

9.- Redireccionamiento.

Para redireccionar hacia otra página (especialmente útil en una sentencia condicional cuando tenemos un script que hace de controlador), solo hay que utilizar el código siguiente:

```
<?php
        header('Location: mipagina.php');
?>
```

Donde mipagina representa la dirección de la página hacia la que se quiere redireccionar. Esta dirección puede ser absoluta y puede tener parámetros de la forma: mipagina.php?param1=val1&param2=val2).
Teóricamente, es mejor preferir una ruta absoluta desde la raíz del servidor (DocumentRoot), configurado en el fichero ***httpd.conf***, de la forma siguiente:

```
<?php
        header('Location: /repositorio/mipagina.php');
?>
```

Si la página de destino estuviera en otro servidor, entonces indicar la URL completa, de la forma siguiente:

```
<?php
        header('Location: http://www.miweb.net/forum/');
?>
```

Encabezados HTTP

Las redirecciones son encabezados HTTP. Pero, según el protocolo HTTP, los encabezados HTTP deben ser enviados antes que cualquier otro tipo de contenido, lo que significa que ningún carácter debe ser enviado (*echo*) antes dela llamada la función header.
En otras palabras, si en el script existe una función header(), no se visualizarán los *echo* previos.

Ejemplo:

```
<body>
<?php
        echo "Esta es la página index.php <br>"; //no lo saca nunca
       /*La función header() es priritaria ante  cualquier código HTML.*/
        $var=7;
        if($var<5){
          header ("Location: pagina1.php");
        }
        else{
          header ("Location: pagina2.php");
}
 ?>
```

```
-------------------------------------------------- pagina1.php------
<?php
       echo "Esta es la página 1";
?>
-------------------------------------------------- pagina2.php------

<?php
      echo "Esta es la página 2";
?>
```

Puede darse el caso de que no tengamos un control en la página de destino para redirigir (tipo href), en tal caso volvería al script del que partió a no ser que incluyamos exit tras el header();

Otra vía de exploración.-

Podemos configurar la llamada en cuanto al retorno del siguiente modo:

```
<?php
// 301 Moved Permanently
header("Location: /foo.php",TRUE,301);

// 302 Found
header("Location: /foo.php",TRUE,302);
header("Location: /foo.php");

// 303 See Other
header("Location: /foo.php",TRUE,303);

// 307 Temporary Redirect
header("Location: /foo.php",TRUE,307);
?>
```

Pese a las pruebas realizadas, no ha funcionado

10.- Implementación de Objetos en PHP.

10.1.- Introducción.

A partir de PHP 5, el modelo de objetos ha sido reescrito para permitir un mejor rendimiento y con más características. Este fue un cambio importante a partir de PHP 4. PHP 5 tiene un modelo de objetos completo.

Entre las características de PHP 5 están la inclusión de la visibilidad, las clases abstractas, clases y métodos finales, interfaces y clonación. De lo que sigue careciendo es de la característica de sobrecarga de métodos y constructores al ser un lenguaje no tipado, esto hace que no se pueda implementar una de las características fundamentales del paradigma orientado a objetos como es el polimorfismo.
PHP trata los objetos de manera referencial, lo que significa que cada variable contiene una referencia (dirección)del objeto propiamente dicho que se encuentra en memoria. En un exceso verbal solemos denominar a la referencia o variable como objeto.

10.2.- class.

La definición básica de clases comienza con la palabra clave **class**, seguido por un nombre de clase, continuado por un par de llaves que encierran las definiciones de las **propiedades** y **métodos** pertenecientes a la clase.

El nombre de clase puede ser cualquier etiqueta válida que no sea una palabra reservada de PHP. Un nombre válido de clase comienza con una letra o un guión bajo, seguido de la cantidad de letras, números o guiones bajos que sea. Como una expresión regular, se expresaría de la siguiente forma: [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*.

Una clase puede tener sus propias constantes, variables (llamadas "propiedades"), y funciones (llamadas "métodos").

10.3.- Propiedades.

Las variables pertenecientes a clases se llaman "propiedades". También se les puede llamar usando otros términos como "atributos" o "campos", pero para PHP se suele utilizar "propiedades". Éstas se definen usando una de las palabras clave **public**, **protected**, o **private**, seguido de una declaración normal de variable. Esta declaración puede incluir una inicialización, pero esta inicialización debe ser un valor constante, es decir, debe poder ser evaluada en tiempo de compilación y no debe depender de información en tiempo de ejecución para ser evaluada.

Si se declara una propiedad utilizando **var** en lugar de public, protected, o private, PHP tratará dicha propiedad como si hubiera sido definida como public. Esta opción está ya en desuso.

Dentro de los métodos de una clase, las propiedades no estáticas pueden ser accedidas utilizando -> (el operador de objeto),**$this->propiedad** (donde propiedad es el nombre de la propiedad). Las propiedades estáticas pueden ser accedidas utilizando **::**, **self::$propiedad**.
La pseudo-variable $this está disponible dentro de cualquier método de clase cuando éste es invocado dentro del contexto de un objeto. $this es una referencia al objeto que se invoca.

10.4.- Definir Clases.

Se hace sin determinar acceso, con la palabra reservada **class**.

Ejemplo utilizando un constructor, métodos setter/getter para encapsulación y el método __toString() :

```
class Caja {
   private $altura;
   private $anchura;
   private $profundidad;
   private $color;
   private $material;
      // constructor
   function __construct($altura, $anchura, $profundidad, $color, $material) {
      $this->setAltura($altura);
      $this->setAnchura($anchura);
      $this->setProfundidad($profundidad);
      $this->setColor($color);
      $this->setMaterial($material);
}
     //métodos
function setAltura($altura) {
       //control $altura
       $this->altura = $altura;
   }
function getAltura() {
       return $this->altura;
   }
function setAnchura($anchura) {
     //control $anchura
      $this->anchura = $anchura;
   }

   function getAnchura() {
      return $this->anchura;
}
   function setProfundidad($profundidad) {
       //control $profundidad
      $this->profundidad = $profundidad;
}
   function getProfundidad() {
      return $this->profundidad;
   }
```

```
   function setColor($color) {
        //control $color
      $this->color = $color;
   }
   function getColor() {
      return $this->color;
   }
   public function setMaterial($material) {
        //control $material
      $this->material = $material;
   }
   public function getMaterial() {
      return $this->material;
   }
   public function __toString() {
    return "Altura :".$this->altura."   Anchura: ".$this->anchura.
         "   Profundidad :".$this->profundidad."
         Color:".$this->color."   Material : ".$this->material."<br>";
   }
}
```

10.5.- Constructores y destructores.

Constructor.

void function __construct ([mixed $args [, $...]])
Método para construir objetos.

Nota: Constructores parent no son llamados implícitamente si la clase child define un constructor. Para ejecutar un constructor parent, se requiere invocar a parent::__construct() desde el constructor child.
Si el child no define un constructor, entonces se puede heredar de la clase padre como un método de clase normal (si no fue declarada como privada).

```
<?php
        class BaseClass {
          function __construct() {
              print "En el constructor BaseClass <br>";
       }
        }

        class SubClass extends BaseClass {
          function __construct() {
             parent::__construct();
             print "En el constructor SubClass<br>";
          }
        }
        class OtherSubClass extends BaseClass {
        }
        $obj = new BaseClass();
        $obj = new SubClass();
        $obj = new OtherSubClass();
?>
```

Salida:

```
En el constructor BaseClass
En el constructor BaseClass
En el constructor SubClass
En el constructor BaseClass
```

Por motivos de compatibilidad, si no está definida la función __construct() para una determinada clase y la clase no heredó uno de una clase padre, buscará el anterior estilo de la función constructora, mediante el nombre de la clase.

Lamentablemente, como ya se ha apuntado, **PHP no permite la sobreescritura de métodos, por tanto, tampoco de constructores**. Es uno de puntos que va en detrimento de PHP frente a JSP cuando se diseña un servidor web orientado a objetos.
El motivo es que las funciones en PHP en general no tienen un control estricto sobre el paso de parámetros como ya vimos anteriormente. Podemos invocar una función con menos parámetros de los requeridos y será un warning, o con más y no dará ningún mensaje de error.

Destructor.

void function __destruct (void)
PHP introduce un concepto de destructor similar al de otros lenguajes orientados a objetos, tal como C++. El método destructor será llamado automáticamente cuando no existan otras referencias a un objeto determinado, o en cualquier otra circunstancia de finalización (fin de script, por ejemplo).

Ejemplo de invocación explícita de un Destructor.

```
<?php
class MyDestructableClass {
function __construct() {
print "En el constructor<br>";
$this->name = "MyDestructableClass";
   }
   function __destruct() {
      print "Destruyendo " . $this->name . "<br>";
}
}
$obj = new MyDestructableClass();
?>
```

Como los constructores, los destructores padre no serán llamados implícitamente por el motor. Para ejecutar un destructor padre, se deberá llamar explícitamente a parent::__destruct() en el interior del destructor. También como los constructores, una clase child puede heredar el destructor de los padres si no implementa uno propio.
El destructor será invocado aún si la ejecución del script es detenida usando exit(). Llamar a exit() en un destructor evitará que se ejecuten las rutinas restantes de finalización. No es una función muy utilizada.

10.6.- Visibilidad.

La visibilidad de una propiedad o método se puede definir anteponiendo una de las palabras claves **public**, **protected** o **private** en la declaración. Miembros de clases declarados como public pueden ser accedidos desde cualquier script que use la clase. Los miembros declarados como protected, solo pueden accederse desde la clase misma y por herencia de la clase parent. Aquellos definidos como private, únicamente por la clase que los definió.

Visibilidad de Propiedades

Las propiedades de clases deben ser definidas como public, private, o protected. Si se declaran usando var, serán definidas como public.

Ejemplo: Declaración de propiedades

```
<?php

        class MyClass {
           public $public = 'Public';
           protected $protected = 'Protected';
           private $private = 'Private';

           function printHello(){
              echo $this->public."<br>";
              echo $this->protected."<br>";
              echo $this->private."<br>";
           }
        }

        $obj = new MyClass();
        echo $obj->public."<br>";
        echo $obj->protected;
       /*Cannot access protected property MyClass::$protected
        in C:\xampp\htdocs\EjemplosPHP\EjemploVisibilidad.php on line 30
        */
        echo $obj->private;
       /*Cannot access private property MyClass::$private
        in C:\xampp\htdocs\EjemplosPHP\EjemploVisibilidad.php on line 32
        */
       $obj->printHello();

        class MyClass2 extends MyClass {
           protected $protected2 = 'Protected2';
           function printHello(){
              echo $this->public."<br>";
              echo $this->protected."<br>";
              echo $this->private;
             /*Undefined property: MyClass2::$private
               in C:\xampp\htdocs\EjemplosPHP\EjemploVisibilidad.php on line 43
              */
              echo $this->protected2."<br>";
           }
        }
```

```
        $obj2 = new MyClass2();
        echo $obj2->public."<br>";
        echo $obj2->private;
       /*Undefined property: MyClass2::$private
         in C:\xampp\htdocs\EjemplosPHP\EjemploVisibilidad.php on line 51
        */
        echo $obj2->protected;
       /*Cannot access protected property MyClass2::$protected
         in C:\xampp\htdocs\EjemplosPHP\EjemploVisibilidad.php on line 53
        */
        echo $this->protected2;
       /*Using $this when not in object context
         in C:\xampp\htdocs\EjemplosPHP\EjemploVisibilidad.php on line 55
       */
        $obj2->printHello();
    ?>
```

Salida al arreglar errores:

```
Public
Public
Protected
Private
Public
Public
Protected
Protected2
```

Visibilidad de Métodos

Los métodos de clases pueden ser definidos como **public**, **private**, o **protected**. Aquellos declarados sin ninguna palabra clave de visibilidad explícita serán definidos como public.

Ejemplo: Declaración de métodos

```
class MyClass
{
   public function __construct() { }
   public function MyPublic() {
         echo "Método Public<br>";
      }
   protected function MyProtected() {
        echo "Método Protected<br>";
      }
   private function MyPrivate() {
         echo "Método Private<br>";
}
   function Foo()
   {
      $this->MyPublic();
      $this->MyProtected();
      $this->MyPrivate();
   }
}

$myclass = new MyClass;
$myclass->MyPublic();
```

```
$myclass->MyProtected();
$myclass->MyPrivate();
$myclass->Foo();
```

Salida:
Error: Call to protected method MyClass::MyProtected() from context '' in C:\xampp\htdocs\PhpProjectBasico\OO_visibilidad_metodos.php on line 30
Error: Call to private method MyClass::MyPrivate() from context '' in C:\xampp\htdocs\PhpProjectBasico\OO_visibilidad_metodos.php on line 31
Al dejar comentadas estas sentencias:
Método Public
Método Public
Método Protected
Método Private

```
class MyClass2 extends MyClass
{
   function Foo2()
   {
      $this->MyPublic();
      $this->MyProtected();
      $this->MyPrivate();
   }
}
$myclass2 = new MyClass2;
$myclass2->MyPublic();
$myclass2->Foo2();
```

Salida script completo:

Error: Call to private method MyClass::MyPrivate() from context 'MyClass2' in C:\xampp\htdocs\PhpProjectBasico\OO_visibilidad_metodos.php on line 47
Al comentar esta sentencia:
Método Public
Método Public
Método Protected
Método Private
Método Public
Método Public
Método Protected

Otro ejemplo:

```
class Bar
{
   public function test() {
      $this->testPrivate();
      $this->testPublic();
   }
   public function testPublic() {
      echo "Bar::testPublic<br>";
   }
   private function testPrivate() {
      echo "Bar::testPrivate<br>";
   }
}
```

```
class Foo extends Bar
{
   public function testPublic() {
      echo "Foo::testPublic<br>";
   }

   private function testPrivate() {
      echo "Foo::testPrivate<br>";
   }
}
$myFoo = new Foo();
$myFoo->test();
```

```
Salida:
Bar::testPrivate
Foo::testPublic
```

10.7.- Herencia de Objetos.

La herencia es una técnica fundamental de la P.O.O. y PHP hace uso de él en su modelado de objetos. Este principio afectará la manera en que muchas clases y objetos se relacionan unas con otras.

Por ejemplo, cuando se extiende una clase, la subclase hereda todas las propiedades y métodos de la clase heredada. A menos que la subclase sobrescriba esos métodos o propiedades, mantendrán su funcionalidad original.

En PHP la herencia es simple y pública (al estilo Java) y se implementa con la palabra reservada **extends**.

Ejemplo basándonos en la anterior declaración de **Caja**.
Admitimos la existencia de una nueva clase **CajaSorpresa** que se define como una caja con un cierto contenido:

```
class CajaSorpresa extends Caja {
   private $contenido;
   // constructor
   public function __construct($_altura, $_anchura, $_profundidad, $_color,
        $_material, $_contenido)
   {
      parent::__construct($_altura, $_anchura, $_profundidad,$_color,$_material);
      $this->setContenido($_contenido);
}

   function getContenido() {
      return $this->contenido;
}

   function setContenido($contenido) {
      $this->contenido = $contenido;
}
```

```
    public function __toString() {
          $cad=parent::__toString();
         $cad.="   Contenido : ".$this->contenido."<br>";
         return $cad;
    }

}
```

Si hacemos un paralelismo con Java, *parent::* es equivalente a *super*.

10.8.- Composición de Objetos.

Para diseñar sitios web cuyas clases no estén relacionadas por herencia, sino por composición o agregación, la solución consiste en definir una propiedad como objeto o array de objetos de otra clase.
Siguiendo con el anterior ejemplo, definimos la clase **Estantería**, que se define una entidad en la que apilaremos las cajas que vayamos creando, una por leja y con capacidad limitada.

```
class Estanteria {
    private $nLejas;
    private $lejasOcupadas=0;
    private $cajas=array();

    function __construct($nLejas) {
        $this->nLejas = $nLejas;
    }

    public function añadirCaja($Caja){
        array_push($this->cajas, $Caja);
        $this->lejasOcupadas++;
}

    public function __toString() {
        $cad="";
        foreach($this->cajas as $posicion=>$contenido){
          $cad.="Leja: ".($posicion+1)." Caja: ".$contenido."<br>";
        }
        return $cad;
    }

}
```

10.9.- Operador de Resolución de Ámbito (::)

El Operador de Resolución de Ámbito (también denominado Paamayim Nekudotayim) o en términos simples, el doble dos-puntos, es un token que permite acceder no solo a los elementos de las clases parent, sino a elementos estáticos, constantes, y **sobrescribir propiedades o métodos de una clase.**
Cuando se hace referencia a estos items desde el exterior de la definición de la clase, se utiliza el nombre de la clase.

A partir de PHP 5.3.0, es posible hacer referencia a una clase usando una variable. El valor de la variable no puede ser una palabra clave (por ej., self, parent y static).

Ejemplo **::** desde el exterior de la definición de la clase

```
<?php
class MyClass {
   const CONST_VALUE = 'Un valor constante';
}

$classname = 'MyClass';
echo $classname::CONST_VALUE ,"<BR>";

echo MyClass::CONST_VALUE ,"<BR>";
?>
```

Salida:
Un valor constante
Un valor constante

Las tres palabras claves especiales **self**, **parent** y **static** son utilizadas para acceder a propiedades y métodos desde el interior de la definición de la clase.

Ejemplo :: desde el interior de la definición de la clase

```
<?php
class OtherClass extends MyClass
{
   public static $my_static = 'variable estática';

   public static function doubleColon() {
      echo parent::CONST_VALUE , “<br>";
      echo self::$my_static , “<br>";
        //accede a elemento estático dentro de la propia clase
   }
}
$classname = 'OtherClass';
echo $classname::doubleColon();
OtherClass::doubleColon();
?>
```

Salida:
Un valor constante
variable estática
Un valor constante
variable estática

Cuando una clase extendida sobrescribe la definición parent de un método, PHP no invocará al método parent. Depende de la clase extendida el hecho de llamar o no al método parent. Esto también se aplica a definiciones de métodos Constructores y Destructores, Sobrecarga, y Mágicos (los reservados por PHP que comienzan por __).

Ejemplo: Invocando a un método parent

```
<?php
      class MyClass
      {
         protected function myFunc() {
            echo "MyClass::myFunc()","<br>";
         }
      }

      class OtherClass extends MyClass
      {
         // Sobrescritura de definición parent
         public function myFunc()
         {
            // Pero todavía se puede llamar a la función parent
            parent::myFunc();
            echo "OtherClass::myFunc()","<br>";
         }
      }

      $class = new OtherClass();
      $class->myFunc();
?>
```

Salida:
MyClass::myFunc()
OtherClass::myFunc()

10.10.- La palabra clave 'static'.

Declarar propiedades o métodos de clases como estáticos los hacen accesibles sin la necesidad de instanciar la clase. Una propiedad declarada como static no puede ser accedida con un objeto de clase instanciado (aunque un método estático sí lo puede hacer).
Por motivos de compatibilidad con PHP 4, si no se utiliza ninguna declaración de visibilidad, se tratará a las propiedades o métodos como si hubiesen sido definidos como public.
Debido a que los métodos estáticos se pueden invocar sin tener creada una instancia del objeto, la pseudo-variable $this no está disponible dentro de los métodos declarados como estáticos. Las propiedades estáticas no pueden ser accedidas a través del objeto utilizando el operador flecha (->).
Invocar métodos no estáticos estáticamente genera una advertencia de nivel E_STRICT.

Como cualquier otra variable estática de PHP, las propiedades estáticas solo pueden ser inicializadas utilizando un string literal o una constante; las expresiones no están permitidas. Por tanto, se puede inicializar una propiedad estática con enteros o arrays (por ejemplo), pero no se puede hacer con otra variable, con el valor de devolución de una función, o con un objeto.

Ejemplo de propiedad estática:

```php
<?php
class Foo
{
   public static $mi_static = 'foo';

   public function valorStatic() {
      return self::$mi_static;
   }
}
class Bar extends Foo
{
   public function fooStatic() {
      return parent::$mi_static;
   }
}
print Foo::$mi_static . "<br>”;
$foo = new Foo();
print $foo->valorStatic() . "<br>";
print $foo->mi_static . "<br>";      // "Propiedad" mi_static no definida
print $foo::$mi_static . "<br>";
$nombreClase = 'Foo';
print $nombreClase::$mi_static . "<br>";
print Bar::$mi_static . "<br>";
$bar = new Bar();
print $bar->fooStatic() . "<br>";
?>
```

```
Salida:
foo
foo
Notice: Accessing static property Foo::$mi_static as non static
in C:\xampp\htdocs\PhpPruebasOO\index.php on line 31
Notice: Undefined property: Foo::$mi_static
in C:\xampp\htdocs\PhpPruebasOO\index.php on line 31
foo
foo
foo
foo
```

Ejemplo de método estático

```php
<?php
class Foo {
   public static function unMétodoEstático() {
// ...
   }
}
Foo::unMétodoEstático();
$nombreClase = 'Foo';
$nombreClase::unMétodoEstático(); // Se puede hacer a partir de PHP 5.3.0
?>
```

10.11.- Abstracción de clases.

A partir de PHP 5 ya se introduce clases y métodos abstractos. Las clases definidas como abstract no son instanciadas y cualquier clase que contiene al menos un método abstracto debe ser definida como abstract. Los métodos definidos como abstractos simplemente declaran la estructura del método, pero no pueden definir la implementación.

Cuando se hereda de una clase abstracta, todos los métodos definidos como abstract en la definición de la clase parent deben ser redefinidos en la clase child; adicionalmente, estos métodos deben ser definidos con la misma visibilidad (o con una menos restrictiva). Por ejemplo, si el método abstracto está definido como protected, la implementación de la función puede ser redefinida como protected o public, pero nunca como private.
Por otra parte, las estructuras de los métodos tienen que coincidir; es decir, la implicación de tipos y el número de argumentos requeridos deben ser los mismos. Por ejemplo, si la clase derivada define un parámetro opcional, mientras que el método abstracto no, no habría conflicto con la estructura del método. Esto también implica a los constructores.

Definir una clase abstracta tiene sentido cuando no vamos a crear objetos de esa clase pero pretendemos obligar a que se implementen ciertos métodos en las clases que hereden de la misma. Es un mecanismo de construcción de interfaces coherentes. En un sentido más amplio, en una jerarquía de clases,

Uno de los grandes hándicaps de PHP respecto a la P.O.O. es que no implementa polimorfismo en toda su extensión, a lo más que llega es a emularlo con clases abstractas.

Ejemplo de clase abstracta

```
abstract class Caja {
   private $altura;
   private $anchura;
   private $profundidad;
   private $color;
   private $material;
        ……….
   abstract function getVolumen();
        ……….
}

class CajaSorpresa extends Caja {
   private $contenido;
        ……….
   public function getVolumen() {
      return $this->getAltura()*$this->getAnchura()*$this->getProfundidad();
}
        …….
}
```

10.12.- Interfaces de objetos.

Las interfaces de objetos permiten crear código con el cual especificamos qué métodos deben ser implementados por una clase, sin tener que definir cómo estos métodos son manipulados.

Las interfaces son definidas utilizando la palabra clave **interface**, de la misma forma que con clases estándar, pero sin métodos que tengan su contenido definido.

Todos los métodos declarados en una interfaz deben ser public, ya que ésta es la naturaleza de una interfaz.

implements

Para implementar una interfaz, se utiliza el operador **implements**. Todos los métodos en una interfaz deben ser implementados dentro de la clase; el no cumplir con esta regla resultará en un error fatal. Las clases pueden implementar más de una interfaz si se deseara, separándolas cada una por una coma.

Nota:

Las interfaces se pueden extender(heredar) al igual que las clases utilizando el operador extends.

Nota:

La clase que implemente una interfaz debe utilizar exactamente las mismas estructuras de métodos que fueron definidos en la interfaz. De no cumplir con esta regla, se generará un error fatal.

Constantes

Es posible tener constantes dentro de las interfaces. Las constantes de interfaces funcionan como las constantes de clases excepto porque no pueden ser sobrescritas por una clase/interfaz que las herede.

Ejemplos.-

Ejemplo de interfaz:

```
<?php

// Declarar la interfaz 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}
```

```
// Implementar la interfaz
class Template implements iTemplate
{
   private $vars = array();

   public function setVariable($name, $var)
   {
      $this->vars[$name] = $var;
   }

   public function getHtml($template)
   {
      foreach($this->vars as $name => $value) {
         $template = str_replace('{' . $name . '}', $value, $template);
      }
      return $template;
   }
}

class BadTemplate implements iTemplate
{
   private $vars = array();

   public function setVariable($name, $var)
        {
      $this->vars[$name] = $var;
   }
}
?>
```

10.13.- Iteración de objetos.

A partir de PHP 5 ya se ofrece una manera de definición de objetos para que sea posible iterar a través de una lista de elementos, con, por ejemplo, una sentencia foreach. Por defecto, todas las propiedades visibles serán utilizadas para la iteración.

Ejemplo Iteración simple de objeto

```
<?php
class MyClass
{
   public $var1 = 'value 1';
   public $var2 = 'value 2';
   public $var3 = 'value 3';

   protected $protected = 'protected var';
   private   $private   = 'private var';

   function iterateVisible() {
      foreach($this as $key => $value) {
         print "$key => $value<br>";
      }
   }
}

$class = new MyClass();

foreach($class as $key => $value) {
   print "$key => $value<br>";
}
//desde fuera de la clase el foreach solo extrae las propiedades accesibles (public)
echo "<br>";
echo "MyClass::iterateVisible:<br>";
$class->iterateVisible();
// al invocar al método interno, extrae todas las propiedades.
?>
```

```
Salida:
var1 => value 1
var2 => value 2
var3 => value 3

MyClass::iterateVisible:
var1 => value 1
var2 => value 2
var3 => value 3
protected => protected var
private => private var
```

Para dar un paso más, se puede implementar la interfaz Iterator. Esto permite al objeto decidir cómo será iterado y qué valores estarán disponibles en cada iteración.

La interfaz Iterator

Introducción
Interfaz para iteradores externos u objetos que pueden ser iterados internamente por sí mismos.
Sinopsis de la Interfaz

```
Iterator extends Traversable {
   /* Métodos */
   abstract public mixed current ( void )
   abstract public scalar key ( void )
   abstract public void next ( void )
   abstract public void rewind ( void )
   abstract public boolean valid ( void )
}
```

Iteradores Predefinidos
PHP ya ofrece un número de iteradores para muchas de las tareas del día a día. Véase la lista de iteradores SPL.

Ejemplo #1 Uso básico
Este ejemplo muestra el orden en el que se llaman a los métodos cuando se emplea un foreach con un iterator.

```
<?php
class myIterator implements Iterator {
   private $position = 0;
   private $array = array(
      "firstelement",
      "secondelement",
      "lastelement",
   );
   public function __construct() {
      $this->position = 0;
   }

   function rewind() {
      var_dump(__METHOD__);
      $this->position = 0;
   }

   function current() {
      var_dump(__METHOD__);
      return $this->array[$this->position];
   }
   function key() {
      var_dump(__METHOD__);
      return $this->position;
   }

   function next() {
      var_dump(__METHOD__);
      ++$this->position;
   }
```

```
    function valid() {
        var_dump(__METHOD__);
        return isset($this->array[$this->position]);
    }
}

$it = new myIterator;
foreach($it as $key => $value) {
var_dump($key, $value);
    echo "<br>";
}
?>
```

Ejemplo #2Iteración de objeto implementando Iterator

```
<?php
class MyIterator implements Iterator
{
    private $var = array();
    public function __construct($array)
    {
        if (is_array($array)) {
            $this->var = $array;
        }
    }
    public function rewind()
    {
        echo "rewinding<br>";
        reset($this->var);
    }
    public function current()
    {
        $var = current($this->var);
        echo "current: $var<br>";
        return $var;
    }
    public function key()
    {
        $var = key($this->var);
        echo "key: $var<br>";
        return $var;
    }
    public function next()
    {
        $var = next($this->var);
        echo "next: $var<br>";
        return $var;
    }
    public function valid()
    {
        $key = key($this->var);
        $var = ($key !== NULL && $key !== FALSE);
        echo "valid: $var<br>";
        return $var;
    }
}
```

```
$values = array(1,2,3);
$it = new MyIterator($values);

foreach ($it as $a => $b) {
print "$a: $b<br>";
}
?>
```

10.14.- Palabra clave 'final'.

A partir de la versión 5, PHP introduce la nueva palabra clave final, que impide que las clases hijas sobrescriban un método, antecediendo su definición con final. No se puede sobreescribir, pero la clases que heredan **sí** pueden usar esos métodos.
Si la propia clase se define como final, entonces no se podrá heredar de ella pero sí crear objetos.

Ejemplo #1 Ejemplo de métodos Final

```
<?php
class BaseClass {
  public function test() {
        echo "llamada a BaseClass::test()<br>";
}

  final public function moreTesting() {
     echo "llamada a BaseClass::moreTesting()<br>";
}
}
class ChildClass extends BaseClass {
public function moreTesting() {
     echo "llamada a ChildClass::moreTesting()<br>";
  }
}
BaseClass::moreTesting()
?>
```

Ejemplo #2 Ejemplo de clase Final

```
<?php
final class BaseClass {
  public function test() {
        echo "llamada a BaseClass::test()<br>";
}

  final public function moreTesting() {
     echo "llamada a BaseClass::moreTesting()<br>";
  }
}
class ChildClass extends BaseClass {
}
 (BaseClass)
?>
```

Nota: Las propiedades no pueden declararse como final. Solo pueden las clases y los métodos.

10.15.- Clonación de Objetos.

La clonación de un objeto consiste en replicar sus atributos en otro espacio de memoria. Hecho que no se consigue si asignamos una variable objeto a otra. Puede ser muy útil cuando pasamos un objeto a un método y no deseamos que sus propiedades se vean afectadas por las operaciones en este método.
Para crear una copia de un objeto se utiliza la palabra clave **clone** (que invoca, si fuera posible, al método **__clone()** del objeto). No se puede llamar al método__clone() de un objeto directamente.

~~$copia_de_objeto = clone $objeto;~~

Cuando se clona un objeto, PHP5 llevará a cabo una copia superficial de las propiedades del objeto. Las propiedades que sean referencias a otras variables, mantendrán las referencias.

void **__clone** (void)
Una vez que la clonación ha finalizado, se llamará al método __clone() del nuevo objeto (si el método __clone() estuviera definido), para permitirle realizar los cambios necesarios sobre sus propiedades.
Ejemplo #1 Clonación de un objeto

```
<?php
class SubObject
{
   static $instances = 0;
   public $instance;

   public function __construct() {
      $this->instance = ++self::$instances;
   }

   public function __clone() {
      $this->instance = ++self::$instances;
   }
}
class MyCloneable
{
   public $object1;
   public $object2;

   function __clone()
   {
      // Forzamos la copia de this->object, si no
      // hará referencia al mismo objeto.
      $this->object1 = clone $this->object1;
   }
}

$obj = new MyCloneable();
$obj->object1 = new SubObject();
```

```
$obj->object2 = new SubObject();

$obj2 = clone $obj;

print("Objeto Original:<br>");
print_r($obj);

print("Objeto Clonado:<br>");
print_r($obj2);

?>
```

El resultado del ejemplo sería:

```
Objeto Original:
MyCloneable Object
(
   [object1] => SubObject Object
      (
         [instance] => 1
      )
   [object2] => SubObject Object
      (
         [instance] => 2
)
)

Objeto Clonado:
MyCloneable Object
(
   [object1] => SubObject Object
      (
         [instance] => 3
      )
   [object2] => SubObject Object
      (
         [instance] => 2
)
)
```

10.16.- Comparación de Objetos.

Al utilizar el operador de comparación (==), se comparan de una forma sencilla las variables de cada objeto, es decir: Dos instancias de un objeto son iguales si tienen los mismos atributos y valores, y son instancias de la misma clase.
Por otra parte, cuando se utiliza el operador identidad (===), las variables de un objeto son idénticas sí y sólo sí hacen referencia a la misma instancia de la misma clase.

Ejemplo de comparación de objetos a partir de PHP 5

```
<?php
function bool2str($bool)
{
   if ($bool === false) {
      return 'FALSO';
   } else {
      return 'VERDADERO';
   }
}
function compararObjetos(&$o1, &$o2)
{
   echo 'o1 == o2 : ' . bool2str($o1 == $o2) . "<br>";
   echo 'o1 != o2 : ' . bool2str($o1 != $o2) . "<br>";
   echo 'o1 === o2 : ' . bool2str($o1 === $o2) . "<br>";
   echo 'o1 !== o2 : ' . bool2str($o1 !== $o2) . "<br>";
}

class Bandera
{
   public $bandera;

   function Bandera($bandera = true) {
      $this->bandera = $bandera;
   }
}

class OtraBandera
{
   public $bandera;

   function OtraBandera($bandera = true) {
      $this->bandera = $bandera;
   }
}

$o = new Bandera();
$p = new Bandera();
$q = $o;
$r = new OtraBandera();

echo "Dos instancias de la misma clase<br>";
compararObjetos($o, $p);

echo "<br>Dos referencias a la misma instancia<br>";
compararObjetos($o, $q);

echo "<br>Instancias de dos clases diferentes<br>";
compararObjetos($o, $r);
?>
```

10.17.- Objetos y referencias.

Uno de los puntos clave de la POO a partir de PHP 5 que a menudo se menciona es que "por omisión los objetos se pasan por referencia". Esto no es completamente cierto. En realidad los objetos se pasan por dirección, ya que el identificador del objeto es una dirección de memoria que contiene el objeto físico. Desde PHP 5, una variable de tipo objeto ya no contiene el objeto en sí como valor. Únicamente contiene un identificador (dirección) del objeto que le permite localizar al objeto real. Cuando se pasa un objeto como parámetro, o se devuelve como retorno, o se asigna a otra variable, las distintas variables no son alias: guardan una copia del identificador, que apunta al mismo objeto.
En PHP, las referencias son “alias” que permiten que 2 variables distintas puedan escribir sobre un mismo valor. O visto de otra manera, son un mecanismo que nos permiten acceder un mismo valor desde nombre de variables distintos y que se comporten como si fueran la misma variable. Hay que tener en cuenta que, en PHP, el nombre de la variable y el contenido de esa variable son 2 cosas totalmente distintas que se enlazan en lo que se llama la tabla de símbolos. De forma que cuando creamos una referencia, simplemente se está añadiendo un alias de dicha variable en la tabla de símbolos de PHP. Veámoslo con un ejemplo, supongamos que tenemos el siguiente código:

```
$a = new Foo();
```

Cuando se ejecuta la instrucción anterior, realmente lo que está sucediendo es que se crea una variable “a” en memoria, se crea un objeto de tipo Foo en memoria y se añade una entrada en la tabla de símbolos de PHP en la que se indica que la variable $a “referencia” (o se relaciona, o “apunta”, o como se quiera llamar) al objeto Foo, pero NO es un puntero a dicho objeto. Si a continuación ejecutamos la instrucción:

```
$b = $a;
```

Lo que ocurre no es que $b se convierta en una referencia de $a, tampoco se puede decir que $b sea una copia de $a. Lo que realmente ha sucedido es que se ha creado una nueva variable “b” en memoria y luego se ha añadido una nueva entrada en la tabla de símbolos indicando que la variable $b, también “referencia” al **mismo** objeto Foo que $a. Si a continuación ejecutamos la instrucción:

```
$c = &$a;
```

Lo que ocurre aquí es que en memoria se habrá creado una tercera variable “c” pero NO se añade una nueva entrada en la tabla de símbolos para “c”, sino que en la tabla de símbolos se indica que “c” es un **alias** de “a”, por lo tanto se comportará de forma idéntica que ésta.

Ejemplo más completo:

```
<?php

class myClass {
public $var;

   function __construct() {
        $this->var = 1;
   }

   function inc() { return ++$this->var; }
}

$a = new myClass();
$b = $a;
echo "\$a = ";var_dump($a);
                // var_dump (), similar a print_r(), imprime detalles de la variable.
echo "\$b = ";var_dump($b);
$c = &$a;
echo "\$c = ";var_dump($c);

$a = NULL;
echo "\$a = ";var_dump($a);
echo "\$b = ";var_dump($b);
echo "\$c = ";var_dump($c);
echo "\$b->var: ".$b->inc();
echo "\$b->var: ".$b->inc();
$b = NULL;
echo "\$b = ";var_dump($b);
```

Ejemplo 2 Referencias y Objetos.

```
<?php
class A {
   public $foo = 1;
}

$a = new A;
$b = $a;
 $b->foo = 2;
echo $a->foo."<br>";
$c = new A;
$d = &$c;
$d->foo = 2;
echo $c->foo."<br>";
$e = new A;
function foo($obj) {
   $obj->foo = 2;
}

foo($e);
echo $e->foo."<br>";
?>
```

10.18.- Excepciones en clases.

PHP 5 tiene un modelo de excepciones similar al de otros lenguajes de programación. Una excepción puede ser lanzada (*thrown*), y atrapada (***catch)*** dentro de PHP. El código puede estar dentro de un bloque ***try***, para facilitar la captura de excepciones potenciales. Cada bloque ***try*** debe tener al menos un bloque ***catch*** correspondiente. Se pueden usar múltiples bloques ***catch*** para atrapar diferentes clases de excepciones. La ejecución normal (cuando no es lanzada ninguna excepción dentro del bloque ***try***, o cuando un bloque ***catch*** que coincide con la clase de la excepción lanzada no está presente) continuará después del último bloque ***catch*** definido en la secuencia. Las excepciones pueden ser ***lanzadas*** (o relanzadas) dentro de un bloque ***catch***.
Cuando una excepción es lanzada, el código siguiente a la declaración no será ejecutado, y PHP intentará encontrar el primer bloque ***catch*** coincidente.
Si una excepción no es capturada, se emitirá un Error Fatal de PHP con un mensaje "***Uncaught Exception ...***" ("Excepción No Capturada"), a menos que se haya definido un gestor con set_exception_handler().

*En PHP 5.5 y posteriores, se puede utilizar un bloque **finally** después de los bloques **catch**. El código de dentro del bloque **finally** siempre se ejecutará después de los bloques **try** y **catch**, independientemente de que se haya lanzado una excepción o no, y antes de que el flujo normal de ejecución continúe.*

El objeto lanzado debe ser una instancia de la clase Exception o de una subclase de Exception. Intentar lanzar un objeto que no lo es resultará en un Error Fatal de PHP.

Nota:
Las funciones internas de PHP utilizan principalmente Información de Errores, sólo las extensiones Orientadas a objetos modernas utilizan excepciones. Sin embargo, los errores se pueden traducir a excepciones simplemente con ErrorException.

Sugerencia
Standard PHP Library (SPL) - (Biblioteca PHP Estándar) proporciona un buen número de excepciones internas.

Ejemplo #1 Lanzar una Excepción

```
<?php
function inverso($x) {
   if (!$x) {
      throw new Exception('División por cero.');
   }
   return 1/$x;
}
```

```php
try {
   echo inverso(5) . "<BR>";
   echo inverso(0) . "<BR>";
} catch (Exception $e) {
   echo 'Excepción capturada: ',  $e->getMessage(), "<BR>";
}
// Continuar la ejecución
echo 'Hola Mundo'.'<BR>';
?>
```

```
El resultado del ejemplo sería:
0.2
Excepción capturada: División por cero.
Hola Mundo
```

Ejemplo #2 Manejo de excepciones con un bloque *finally*
[No admitida en PHP 5.4]

```php
<?php
function inverse($x) {
   if (!$x) {
      throw new Exception('División por cero.');
   }
   return 1/$x;
}

try {
   echo inverse(5) . "<br>";
} catch (Exception $e) {
   echo 'Excepción capturada: ',  $e->getMessage(), "<br>";
} finally {
   echo "Primer finally.<br>";
}

try {
   echo inverse(0) . "<br>";
} catch (Exception $e) {
   echo 'Excepción capturada: ',  $e->getMessage(), "<br>";
} finally {
   echo "Segundo finally.<br>";
}
// Continuar ejecución
echo 'Hola Mundo<br>';
?>
```

```
El resultado del ejemplo sería:
0.2
Primer finally.
Excepción capturada: División por cero.
Segundo finally.
Hola Mundo
```

Ejemplo #3 Excepciones Anidadas

```
<?php
class MiExcepción extends Exception { }
class Prueba {
   public function probar() {
      try {
         try {
            throw new MiExcepción('foo!');
         } catch (MiExcepción $e) {
            // relanzarla
            throw $e;
         }
      } catch (Exception $e) {
         var_dump($e->getMessage());
      }
   }
}

$foo = new Prueba;
$foo->probar();
?>
```

El resultado del ejemplo sería:

```
string(4) "foo!"
```

10.19.- Excepciones propias.

Una clase de Excepción definida por el usuario puede ser definida ampliando la clase Exception interna. Los miembros y las propiedades de abajo muestran lo que es accesible dentro de la clase hija que deriva de la clase Exception interna.

Ejemplo #1 La clase Exception Interna

```
<?php
class Exception
{
   protected $message = 'Unknown exception';   // mensaje de excepción
   private   $string;                          // caché de __toString
   protected $code = 0;                        // código de excepción definido por el usuario
   protected $file;                            // nombre de archivo fuente de la excepción
   protected $line;                            // línea fuente de la excepción
   private   $trace;                           // determinación del origen
   private   $previous;                        // excepción previa si la excepción está anidada

   public function __construct
($message = null, $code = 0, Exception $previous = null);

   final private function __clone(); // Inhibe la clonación de excepciones.

   final public  function getMessage();       // mensaje de excepción
   final public  function getCode();          // código de excepción
   final public  function getFile();          // nombre de archivo fuente
```

```
    final public  function getLine();          // línea fuente
    final public  function getTrace();         // un array de backtrace()
    final public  function getPrevious();      // excepción anterior
    final public  function getTraceAsString();
// string formateado del seguimiento del origen. Sobrescribible
    public function __toString();    // string formateado para mostrar
}
?>
```

Si una clase extiende la clase Exception interna y redefine el constructor, se recomienda que también llame a parent::__construct() para asegurarse que toda la información disponible haya sido asignada apropiadamente. El método __toString() puede ser sobrescrito para proporcionar una salida personalizada cuando el objeto es presentado como un string.

Nota:
Las excepciones no se pueden clonar. Intentar clonar una Excepción resultará en un error **E_ERROR** fatal.

Ejemplo #2 Extender la clase Exception

```
<?php
/**
 * Definir una clase de excepción personalizada
 */
class MiExcepción extends Exception
{
    // Redefinir la excepción, por lo que el mensaje no es opcional
    public function __construct
($message, $code = 0, Exception $previous = null) {
        //  código

        // asegúrate de que todo está asignado apropiadamente
        parent::__construct($message, $code, $previous);
    }

    // representación de cadena personalizada del objeto
    public function __toString() {
        return __CLASS__ . ": [{$this->code}]: {$this->message}<br>";
          // __CLASS__  ,nombre de la clase:  MiExcepción
    }

    public function funciónPersonalizada() {
        echo "Una función personalizada para este tipo de excepción<br>";
    }
}
?>
```

Cuando un método lanza una excepción propia y no la captura, en PHP no está obligado a avisar con *throws Exception*.

[Proponer ejercicio de aplicación O.O. Cajas]

11.- Manejo de datos en el servidor.

El sistema de acceso y manipulación de bases de datos desde PHP es similar al de otros lenguajes de script: establece la conexión con la base de datos, ejecuta las sentencias de consulta o modificación y finalmente cierra la conexión.
PHP soporta compatibilidad con accesos a múltiples sistemas de bases de datos, sin embargo, el modo de programación sobre cada tipo de base de datos, no presenta, como en otros casos, la misma nomenclatura. PHP utiliza funciones de nombre genérico, pero precedidas normalmente por el nombre del sistema de base de datos, de modo que las funciones que ofrece el lenguaje para el acceso a cada tipo de base de datos son diferentes.

Hasta la versión 5 de PHP, el acceso a las bases de datos se hacía principalmente utilizando extensiones específicas para cada sistema gestor de base de datos (extensiones nativas).
A partir de la versión 5 de PHP se introdujo en el lenguaje una extensión para acceder de una forma común a distintos sistemas gestores: PDO. La gran ventaja de PDO está clara: podemos seguir utilizando una misma sintaxis aunque cambiemos el motor de nuestra base de datos. Por el contrario, en algunas ocasiones preferiremos seguir usando extensiones nativas en nuestros programas. Mientras PDO ofrece un conjunto común de funciones, las extensiones nativas normalmente ofrecen más potencia (acceso a funciones específicas de cada gestor de base de datos) y en algunos casos también mayor velocidad.

11.1.- Extensión MySQLi.

Las opciones de configuración de PHP se almacenan en el fichero ***php.ini***.
En este fichero hay una sección específica para las opciones de configuración propias de cada extensión. Entre las opciones que puedes configurar para la extensión MySQLi están:

mysqli.allow_persistent. Permite crear conexiones persistentes.
mysqli.default_port. Número de puerto TCP predeterminado a utilizar cuando se conecta al servidor de base de datos.
mysqli.reconnect. Indica si se debe volver a conectar automáticamente en caso de que se pierda la conexión.
mysqli.default_host. Host predeterminado a usar cuando se conecta al servidor de base de datos.
mysqli.default_user. Nombre de usuario predeterminado a usar cuando se conecta al servidor de base de datos.
mysqli.default_pw. Contraseña predeterminada a usar cuando se conecta al servidor de base de datos.

11.1.1.- Establecimiento de conexión con bases de datos.

Ofrece un interface de programación dual, pudiendo accederse a las funcionalidades de la extensión utilizando objetos o funciones de forma indiferente. Por ejemplo, para establecer una conexión con un servidor MySQL y consultar su versión, podemos utilizar cualquiera de las siguientes formas:

```
// utilizando constructores y métodos de la programación orientada a objetos
$conexion = new mysqli('localhost', 'usuario', 'contraseña', 'base_de_datos');
print $conexion->server_info;

// utilizando llamadas a funciones
$conexion = mysqli_connect('localhost', 'usuario', 'contraseña', 'base_de_datos');
print mysqli_get_server_info($conexion);
```

En ambos casos, la variable **$conexion** es de tipo objeto. La utilización de los métodos y propiedades que aporta la clase mysqli normalmente produce un código más corto y legible que si utilizamos llamadas a funciones.
Es importante verificar que la conexión se ha establecido correctamente. Para comprobar el error, en caso de que se produzca, se pueden usar las siguientes propiedades de la clase mysqli:

- **connect_errno** , devuelve el **número de error** ó 0 (null) si no se produce ningún error.

  connect_errno:
  Es 0 cuando no hay error.
  1045 cuando el usuario o clave no es correcto.
  2002 cuando el server no es correcto.

- **connect_error** , devuelve el **mensaje de error** o null si no se produce ningún error.

Por ejemplo, el siguiente código comprueba el establecimiento de una conexión con la base de datos "bd_empleados" y finaliza la ejecución si se produce algún error:

```
@ $conexion = new mysqli('localhost', 'root', 'root', ‘bd_empleados');
$error = $conexion->connect_errno;
if ($error != null) {
        echo "<p>Error $error conectando a la base de datos:
                $conexion->connect_error</p>";
exit();
}
```

En PHP, como veremos posteriormente con más detalle, puedes anteponer a cualquier expresión el operador de control de errores @ para que se ignore cualquier posible error interno que pueda producirse al ejecutarla y que sea el programador el que tome el control de los errores.

Si una vez establecida la conexión, queremos cambiar la base de datos se puede usar el método **select_db()** para indicar el nombre de la nueva.

```
// utilizando el método connect
$conexion ->select_db('otra_bd');
```

Una vez finalizadas las tareas con la base de datos, se utiliza el método **close** para cerrar la conexión con la base de datos y liberar los recursos que utiliza.

```
$conexion ->close();
```

11.1.2.- Operaciones sobre bases de datos MySQL

query()

Es la forma más inmediata de ejecutar una consulta.
Si la consulta es de acción que no devuelve datos (como una sentencia SQL de tipo UPDATE, INSERT o DELETE), la llamada devuelve **true** si se ejecuta correctamente o **false** en caso contrario, pero ejecutarse correctamente solo significa que no hay error de sintaxis en la orden SQL, no que ha realizado la operación.
Si esperamos que la operación afecte al menos a una fila, entonces para ver el número de registros afectados se puede obtener con la propiedad **affected_rows.**

```
echo "BORRAR POR NIF <br>";
 $Vnif=$_REQUEST['nif']; //lo recuperamos de un formulario
$orden="DELETE FROM EMPLEADO WHERE NIF='".$Vnif."'";
                                                    //$Vnif es cadena.
 $resultado=$conexion->query($orden);
 echo '$resultado :  '.$resultado."<br>";
 echo "Se ha borrado  ".$conexion->affected_rows. " registro.";

//Cuando existe el NIF
BORRAR POR NIF SELECCIONADO
$resultado : 1
Se ha borrado 1 registro.

//Cuando no existe el NIF
BORRAR POR NIF SELECCIONADO
$resultado : 1
Se ha borrado 0 registro.
```

En el caso de ejecutar una sentencia SQL que sí devuelva datos (como un SELECT), éstos se devuelven en forma de un objeto resultado (de la clase **mysqli_result**).
En este caso, el método query tiene un parámetro opcional que afecta a cómo se obtienen internamente los resultados, pero no a la forma de utilizarlos posteriormente.
En la opción por defecto, **MYSQLI_STORE_RESULT**, los resultados se recuperan todos juntos de la base de datos y se almacenan deforma local. Si cambiamos esta opción por el valor **MYSQLI_USE_RESULT**, los datos se van recuperando del servidor según se vayan necesitando.

```
$orden="SELECT * FROM EMPLEADO WHERE idDepartamento="
                .$VidDepartamento;  //$VidDepartamento es INT.
$resultado=$conexion->query($orden);
```

Los datos se guardan en una caché en forma de array bidimensional cuyo nombre será el asignado en la consulta ($resultado).

Si la consulta es errónea $resultado es false (sintaxis) pero si no devuelve nada la manera de saberlo es utilizando **num_rows**, ya que el objeto $resultado apunta a un array vacío pero no es falso.

Importante: **No son directamente utilizables por scripts de PHP.**
Cuando estos datos ya no los necesitemos, los podemos liberar con el método **free** de la clase**mysqli_result**:

```
$resultado->free();
```

Para recuperar los datos obtenidos del servidor en la zona de memoria intermedia ($resultado) y que sean utilizables por PHP, tenemos varias posibilidades:

▪ **fetch_array()** Obtiene un registro completo del conjunto de resultados y lo almacena en un array. Por defecto el array contiene tanto claves numéricas como asociativas.

```
if($resultado->num_rows>0){
   $fila=$resultado->fetch_array();
      while($fila){
               echo"   NIF: ",$fila['nif'];
               echo "   NOMBRE: ",$fila['nombre'];
               echo "   APELLIDOS: ",$fila['apellidos'];
               echo "   PROFESIÓN: ",$fila['profesion'];
               echo "   SALARIO: ",$fila['salario'];
               echo "   FECHA INGRESO: ",$fila['fechaIng'],"<BR>";
               $fila=$resultado->fetch_array();
      }
}
else{
   print_r("Resultado fallido : ".$resultado->num_rows);
}
```

Este comportamiento por defecto se puede modificar utilizando un parámetro opcional, que puede tomar los siguientes valores:
MYSQLI_NUM. Devuelve un array con claves numéricas.
MYSQLI_ASSOC. Devuelve un array asociativo.
MYSQLI_BOTH. Es el comportamiento por defecto, en el que devuelve un array con claves numéricas y asociativas.

▪ **fetch_assoc()**. Idéntico a fetch_array pasando como parámetro **MYSQLI_ASSOC**.
Si utilizamos esta opción, solo podemos usar índices asociativos.

▪ **fetch_row** (función **mysqli_fetch_row)**. Idéntico a fetch_array pasando como parámetro **MYSQLI_NUM**.
Si utilizamos esta opción, solo podemos usar índices numéricos.

▪ **fetch_object()**. Similar a los métodos anteriores, pero devuelve un objeto en lugar de un array.
Los objetos que extraigamos con fetch_object() los considera del tipo **stdClass**.
Esta clase no tiene ni propiedades, ni métodos, es una clase vacía.
Se puede usar cuando necesitamos un objeto genérico al que luego el podremos añadir propiedades. Ejemplo:

```
<?php
$obj = new stdClass();
        $obj->codigo = "34554356787";
        $obj->nif= "09877876-V";
        $obj->nombre= "Adolfo Aldana”;
?>
```

También se puede usar como molde para crear objetos cuyas propiedades son los datos del resultado del SELECT. Habrá una propiedad por cada columna extraída con el SELECT.
Este objeto puede ser asignado a un identificador.
Ej. $obj=$resultado->fetch_object();
A las propiedades se accede, por ejemplo:
$obj->nombre.

Es un atajo si lo que queremos es escribir en una vista el resultado de una consulta (un listado de empleados por ejemplo), evita crear objetos de tipo Empleado, ya que crea automáticamente objetos de tipo stdClass con las propiedades del SELECT, que serán propiedades de Empleado (puede que no todas).

11.1.3.- Consultas preparadas.

Cada vez que se envía una consulta al servidor, éste debe analizarla antes de ejecutarla. Algunas sentencias SQL, como las que insertan valores en una tabla, deben repetirse de forma habitual en un programa. Para acelerar este proceso, MySQL admite consultas preparadas. Estas consultas se almacenan en el servidor listas para ser ejecutadas cuando sea necesario. MySQL ejecutará la sentencia tantas veces como se quiera cambiando los valores.
Con esta implementación se evita, en lo posible, el **SQLInjection** que es la incrustación de código malicioso en un formulario de petición de datos.

Los pasos que debes seguir para ejecutar una consulta preparada son:

1.- Preparar la consulta en el servidor MySQL utilizando el método **prepare()**.
2.- Invocar al método **bind_param()** que permite tener una consulta preparada en el servidor.

Mediante estas dos instrucciones, el servidor analiza la orden SQL transferida y detecta si hay código malicioso incrustado.

Estilo orientado a objetos:
bool mysqli_stmt->bind_param (string $types , &$var1[, &$...])

Especificación del tipo de caracteres (types)

Carácter	Descripción
i	la variable correspondiente es de tipo entero
d	la variable correspondiente es de tipo double
s	la variable correspondiente es de tipo string
b	la variable correspondiente es un blob y se envía en paquetes

3.- Ejecutar la consulta, tantas veces como sea necesario, con el método **execute**().
bool mysqli_stmt->execute (void)

4.- Una vez que ya no se necesita más, se debe ejecutar el método **close** ().

Ejemplo: Inserciones repetidas utilizando sentencias preparadas
Este ejemplo realiza dos consultas INSERT sustituyendo *name* y *value* por los marcadores posicionales '?'.

```
<?php
$sentencia = $conexion->prepare("INSERT INTO TABLA (name, value)
                                 VALUES (?, ?)");
$sentencia->bind_param("si", $nombre, $valor);
// insertar una fila
$nombre = 'uno';
$valor = 1;
$sentencia->execute();

// insertar otra fila con diferentes valores
$nombre = 'dos';
$valor = 2;
$sentencia->execute();
$sentencia->close();
?>
```

Sentencias preparadas para evitar SQLInjection.

- Uso de sentencias preparadas en inserción de datos:

```
……..
$Sql=$conexion->prepare("INSERT INTO ESTANTERIA
(CODIGO_ESTANTERIA,MATERIAL,NUMERO_LEJAS,PASILLO,NUMERO)
VALUES(?,?,?,?,?)");
$Sql->bind_param("ssiss",$Codigo,$Material,$NumeroLejas,$Pasillo,$Numero);
$resultado=$Sql->execute();
if($resultado==null){
  $Code=$conexion->errno;
  $Message=$conexion->error;
  throw new EstanteriaException($Message,$Code,"INSERT ESTANTERIA");
}
else header("Location:../VISTA/MenuEntrada.php");
……
```

Para que se produzca un ataque en una inserción, el atacante debe conocer al menos parte de la estructura de nuestras tablas.
Ejemplo de inserción de código malicioso en un formulario de consulta o modificación (SQLInjection):
En una operación de lectura o actualización en donde utilicemos una clausula WHERE nuestra base de datos se vuelve más vulnerable.
Por ejemplo, si en un campo de texto (type="text") permitimos cualquier cadena de entrada (lo habitual) y no controlamos la longitud, podremos encontrarnos con el siguiente intento en un formulario que pide nif y password:
Como valor de nif colocamos uno válido pero acompañado de algo más: ***99989677X' or '1'='1*** , es lógico que vaya aplicado a un SELECT con WHERE asociado al nif, por tanto, la comparación podría ser algo así: WHERE nif='*99989677X' or '1'='1' AND password="……".*

Así pues, la consulta siempre será cierta porque 1 es 1 y está relacionado por OR, y aunque insertemos un password erróneo siempre devolverá los datos de la persona cuyo NIF indicamos.

El consecuente error sería aplicar UPDATE O DELETE creyendo que solo afecta a esa fila. Pero la operación anterior también ira condicionada por el WHERE del nif, o sea, que siempre será cierta para todas las filas de la tabla.

Conclusión, con saber el NIF de las personas registradas sabremos todo sus datos. Incluso podremos modificarlos o borrarlos.
Nota: En el formulario, los controles del NIF con HTML5 o Javascript, pueden hacer imposible el SQLInjection, pero no ocurre así en campos de búsqueda por apellidos, nombre o domicilio.

- Ejemplo uso con SELECT.

Para poder aplicar la protección en sentencias SELECT aparecen dos nuevas órdenes:

- mysqli_stmt::bind_result (mixed &$var1 [, mixed &$...]) : bool

y

- mysqli_stmt::get_result (void) : mysqli_result

Ambas están preparadas para extraer resultados de la base de datos.
La primera cuando son conocidas las variables en la claúsula SELECT, y la segunda cuando lo hacemos utilizando *.
get_result() es más versátil, ya que puede utilizarse también para el primer caso. Siempre va relacionada con índices asociativos.

Utilizando bind_result.

```
$ordenSQL="SELECT nif, password FROM votante WHERE nif=? AND
                    ?=AES_DECRYPT(PASSWORD, 'daniel')AND votado='No'";
$SentenciaPreparada=$conexion->prepare($ordenSQL);
$SentenciaPreparada->bind_param("ss",$Nif,$Password);
$estado=$SentenciaPreparada->execute(); //devuelve un booleano (1/0)
```

```
// Ligar el resultado
   $nif = "";
   $password = "";
   $Sentencia->bind_result($nif, $password);

// Recuperar los datos
   while ($Sentencia->fetch()) {
      echo $nif;
      echo $password;
}
   $Sentencia->close();
```

Utilizando get_result().

```
.....
$ordenSQL="SELECT * FROM votante WHERE nif=? AND
                  ?=AES_DECRYPT(PASSWORD, 'daniel')AND votado='No'";
$SentenciaPreparada=$conexion->prepare($ordenSQL);
$SentenciaPreparada->bind_param("ss",$Nif,$Password);
$estado=$SentenciaPreparada->execute(); //devuelve un booleano (1/0)
$resultado=$SentenciaPreparada->get_result();
 if($estado!=null && $resultado->num_rows==1){
        $filaVotante=$resultado->fetch_array();
        ..................
}
```

11.1.4.- Transacciones.

Cuando debemos realizar una serie de consultas SQL, especialmente INSERT, DELETE y UPDATE, y el fallo de alguna de ellas o alguna condición necesaria no se cumpla, debemos volver al estado inicial utilizando transacciones. Un aspecto importante es determinar cuándo una operación falla. Lo más recomendable es averiguar el estado del objeto que ha obtenido la conexión.
Debemos tener en cuenta que si no se expresa lo contrario, cualquier orden SQL que afecte a los datos en las tablas se realiza automáticamente, el commit es inmediato. Para que esto no ocurra y solo al final de un proceso acotado se valide todo, lo que proponemos es guardar los cambios en las tablas en una zona virtual, no real. Desde el punto de vista práctico es como si hiciésemos las operaciones sobre 'copias' de las tablas originales.
Las transacciones se permiten cuando utilizamos tablas InnoDB.

Este comportamiento se gestiona con el método **autocommit.**

```
$conexion->autocommit(false);// deshabilitamos el modo transaccional automático
```

Al deshabilitar las transacciones automáticas, las siguientes operaciones sobre la base de datos iniciarán una transacción que finalizará utilizando:

- **commit.** Realizar una confirmación de la transacción actual, devolviendo true si se ha realizado correctamente o false en caso contrario.
- **rollback.** Realizar una operación vuelta atrás de la transacción actual, devolviendo true si se ha realizado correctamente o false en caso contrario.

Ejemplo:

```
<?php
@ $conexion = new mysqli("localhost", "root", "root", "bd_almacen");
$error = $conexion->connect_errno;
```

```
if ($error != null) {
print "<p>Se ha producido el error: $conexion->connect_error.</p>";
exit();
}

// Definimos una variable para comprobar la ejecución de las consultas
$todo_bien = true;
$conexion->autocommit(false); // Iniciamos la transacción
$sql = 'UPDATE articulo SET unidades=0 WHERE producto="122334xcv";
$resultado=$conexion->query($sql);
if ($conexion-> affected_rows<1)
        //Supongo que estará mal si no ha actualizado ninguna fila
        $todo_bien = false;
$sql = 'INSERT INTO articulo (`producto`, `tienda`, `unidades`)
        VALUES ("3990998tyu", 3,12)';
$resultado=$conexion->query($sql);

if ($conexion-> affected_rows<1)
        //Supongo que estará mal si no ha insertado ninguna fila
        $todo_bien = false;
// Si todo fue bien, confirmamos los cambios
// y en caso contrario los deshacemos
if ($todo_bien == true) {
        $conexion->commit();
        print "<p>Los cambios se han realizado correctamente.</p>";
}else {
        $conexion->rollback();
        print "<p>No se han podido realizar los cambios.</p>";
}
$conexion->close(); //cierra la conexión y automáticamente la transacción.
```

Opciones posteriores:

```
unset($conexion); //destruye la variable de la conexión,  es raro usarlo.
$conexion->autocommit(true); //vuelve a habilitar el autocommit.
?>
```

Nota importante:
Existe siempre la duda de utilizar transacciones o disparadores en acciones que impliquen más de una orden SQL.
Ambas pueden realizar el cometido, pero un disparador no ejecutado por una orden que no genere error por sí misma (UPDATE, DELETE) supone la no detección del mismo.

[Proposición Enunciado Proyecto PHP O. O. Empleados]

12.- Funciones predefinidas.

12.1.- Funciones de tiempo y fecha.

Introducción
Estas funciones permiten obtener la fecha y la hora del servidor donde se están ejecutando los scripts PHP. Se pueden usar estas funciones parar dar formato a fechas y horas de muchas maneras diferentes.
La información de la fecha y de la hora se almacena internamente como un número de 64 bits, por lo que se admiten todas las fechas útiles posibles (incluidos años negativos). El rango va aproximadamente de 292 billones de años en el pasado hasta lo mismo en el futuro.

date
date — Dar formato a la fecha/hora local

string date (string $format [, int $timestamp = time()])
Devuelve una cadena formateada según el formato dado usando el parámetro de tipo integer timestamp dado o el momento actual si no se da una marca de tiempo. En otras palabras, timestamp es opcional y por defecto es el valor de time().

format
El formato de la fecha de salida tipo string. Ver las opciones de formato más abajo. También hay varias constantes de fecha predefinidas que pueden usarse en su lugar, así por ejemplo DATE_RSS contiene la cadena de formato 'D, d M Y H:i:s'.
Los siguientes caracteres están reconocidos en el parámetro de cadena format.

Car.	Descripción	Ejemplo
Día		
d	Día del mes, 2 dígitos con ceros iniciales	1 a 31
D	Una representación textual de un día, tres letras	Mon hasta Sun
j	Día del mes sin ceros iniciales	1 a 31
l	Una representación textual completa del día de la semana	Sunday hasta Saturday
N	Representación numérica ISO-86 1 del día de la semana (añadido en PHP 5.1.0)	1 (para lunes) hasta 7 (para domingo)
S	Sufijo ordinal inglés para el día del mes, 2 caracteres	st, nd, rd o th. Funciona bien con j
w	Representación numérica del día de la semana	0 (para domingo) hasta 6 (para sábado)
z	El día del año (comenzando por 0)	0 hasta 365
Semana		
W	Número de la semana del año ISO-8601, las semanas comienzan en lunes (añadido en PHP 4.1.0)	Ejemplo: 42 (la 42ª semana del año)
Mes		
F	Una representación textual completa de un mes, como January o March	January hasta December

m	Representación numérica de una mes, con ceros iniciales	01 hasta 12
M	Una representación textual corta de un mes, tres letras	Jan hasta Dec
n	Representación numérica de un mes, sin ceros iniciales	1 hasta 12
t	Número de días del mes dado	28 hasta 31
Año		
L	Si es un año bisiesto	1 si es bisiesto,0 si no.
Y	Una representación numérica completa de un año, 4 dígitos	Ejemplos: 1999 o 2 3
y	Una representación de dos dígitos de un año	Ejemplos: 99 o 03
Hora		
a	Ante meridiem y Post meridiem en minúsculas	am o pm
A	Ante meridiem y Post meridiem en mayúsculas	AM o PM
B	Hora Internet	hasta 999
g	Formato de 12 horas de una hora sin ceros iniciales	1 hasta 12
G	Formato de 24 horas de una hora sin ceros iniciales	000 hasta 23
h	Formato de 12 horas de una hora con ceros iniciales	01 hasta 12
H	Formato de 24 horas de una hora con ceros iniciales	00 hasta 23
i	Minutos, con ceros iniciales	00 hasta 59
s	Segundos, con ceros iniciales	00 hasta 59
s	Segundos, con ceros iniciales	00 hasta 59
u	Microsegundos	Ejemplo: 654321
Zona horaria		
e	Identificador de zona horaria (añadido en PHP 5.1.0)	Ejemplos: UTC, GMT, Atlantic/Azores
I	Si la fecha está en horario de verano o no	1 horario de verano, 0 si no.
O	Diferencia de la hora de Greenwich (GMT) en horas	Ejemplo: +0200
P	Diferencia con la hora de Greenwich (GMT) con dos puntos entre horas y minutos (añadido en PHP 5.1.3)	Ejemplo: +02:00
T	Abreviatura de la zona horaria	Ejemplos: EST, MDT ...
Z	Índice de la zona horaria en segundos. El índice para zonas horarias al oeste de UTC siempre es negativo, y para aquellas al este de UTC es siempre positivo.	-43200 hasta 50400
Fecha/Hora Completa		
c	Fecha ISO 8601	2004-02-12T15:19:21+00:00
r	Fecha con formato » RFC 2822	Thu, 21 Dec 2000 16:01:07 +0200
U	Segundos desde la Época Unix (1 de Enero del 1970 00:00:00 GMT)	

Los caracteres no reconocidos en la cadena de formato serán impresos tal cual. El formato Z siempre devolverá cuando se usa gmdate().

Nota:
Ya que esta función sólo acepta marcas de tiempo de tipo integer el carácter de formato u sólo es útil cuando se usa la función date_format() con marcas de tiempo basadas en usuario creadas con date_create().

timestamp

El parámetro opcional timestamp es una marca de tiempo Unix de tipo integer que por defecto es la hora local si no se proporciona ningún valor a timestamp. En otras palabras, es por defecto el valor de la función time().

Devuelve una cadena de fecha formateada. Si se usa un valor no numérico para timestamp, se devuelve FALSE y se emite un error de nivel E_WARNING.

Cada vez que se llame a la función date/time se generará un E_NOTICE si la zona horaria no es válida, y/o un mensaje E_STRICT o E_WARNING si se usa la configuración del sistema o la variable global TZ. Vea también date_default_timezone_set()

Ejemplo #1 Ejemplo de date()

```
<?php
/* Establecer la zona horaria predeterminada a usar. */
date_default_timezone_set('UTC');

// Imprime algo como: Monday
echo date("l");

// Imprime algo como: Monday 8th of August 2005 03:12:46 PM
echo date('l jS \of F Y h:i:s A');

// Imprime: July 1, 2000 is on a Saturday
echo "July 1, 2000 is on a " . date("l", mktime(0, 0, 0, 7, 1, 2000));

/* Usar las constantes en el parámetro de formato */
// Imprime algo como: Wed, 25 Sep 2013 15:28:57 -0700
echo date(DATE_RFC2822);

// Imprime algo como: 2000-07-01T00:00:00+00:00
echo date(DATE_ATOM, mktime(0, 0, 0, 7, 1, 2000));
?>
```

Es posible usar date() y mktime() juntos para buscar fechas en el futuro o en el pasado.

Ejemplo #2 Ejemplo de date() y mktime()

```
<?php
$mañana      = mktime(0, 0, 0, date("m")  , date("d")+1, date("Y"));
$mes_anterior  = mktime(0, 0, 0, date("m")-1, date("d"),   date("Y"));
$año_siguiente = mktime(0, 0, 0, date("m"),   date("d"),   date("Y")+1);
?>
```

Nota:
Esto puede ser más fiable que añadir o sustraer simplemente el número de segundos de un día o mes a una marca de tiempo debido al horario de verano.

Algunos ejemplos de formatear date().

Ejemplo #3 date() Formatting

```
<?php
// Se asume que hoy es March 10th, 2001, 5:16:18 pm, y que estamos en la
// zona horaria Mountain Standard Time (MST)

$hoy = date("F j, Y, g:i a");                 // March 10, 2001, 5:16 pm
$hoy = date("m.d.y");                         // 03.10.01
$hoy = date("j, n, Y");                       // 10, 3, 2001
$hoy = date("Ymd");                           // 20010310
$hoy = date('h-i-s, j-m-y, it is w Day');
                        // 05-16-18, 10-03-01, 1631 1618 6 Satpm01
$hoy = date('\i\t \i\s \t\h\e jS \d\a\y.');   // it is the 10th day.
$hoy = date("D M j G:i:s T Y");               // Sat Mar 10 17:16:18 MST 2001
$hoy = date('H:m:s \m \i\s\ \m\o\n\t\h');     // 17:03:18 m is month
$hoy = date("H:i:s");                         // 17:16:18
$hoy = date("Y-m-d H:i:s");
                        // 2001-03-10 17:16:18 (el formato DATETIME de MySQL)
?>
```

Nota:
Para generar una marca de tiempo desde una cadena que representa la fecha, puedes usar **strtotime()**. Adicionalmente, algunas bases de datos tienen funciones para convertir formatos de fecha en marcas de tiempo (como la función UNIX_TIMESTAMP de MySQL).

Sugerencia
La marca de tiempo del inicio de una petición está disponible en $_SERVER['REQUEST_TIME'] desde PHP 5.1.

time

int time (void)
Devuelve el momento actual medido como el número de segundos desde la Época Unix (1 de Enero de 1970 00:00:00 GMT).

Ejemplo #1 Ejemplo de time()

```
<?php
$semana_sig = time() + (7 * 24 * 60 * 60);
                // 7 días; 24 horas; 60 minutos; 60 segundos
echo 'Ahora:       '. date('Y-m-d') ."\n";
echo 'Semana Siguiente: '. date('Y-m-d', $semana_sig) ."\n";
                // o usar strtotime():
echo 'Semana Siguiente: '. date('Y-m-d', strtotime('+1 week')) ."\n";
?>
```

El resultado del ejemplo sería algo similar a:
Ahora: 2005-03-30
Semana Siguiente: 2005-04-06
Semana Siguiente: 2005-04-06

Sugerencia
La marca de tiempo del inicio de la petición está disponible en $_SERVER['REQUEST_TIME'] desde PHP 5.1.

Ejemplos prácticos.

- Para convertir cadenas en fechas y darles un formato distinto al que pudieran tener:

```
$FechaRegreso_2=date("d-m-Y",strtotime($FechaRegreso_1));
```

Para manipular un dato de tipo Time y quitar los segundos:

```
$HoraSalida_2= date("H:i",strtotime($HoraSalida_1));
```

Sumar minutos a una hora:

```
$HoraNueva =date("H:i",strtotime("+$_minutos minute",strtotime( $_Hora)));
```

- **Calcular días entre dos fechas con PHP.**

```
//defino fecha 1
$ano1 = 2006;
$mes1 = 10;
$dia1 = 2;

//defino fecha 2
$ano2 = 2006;
$mes2 = 10;
$dia2 = 27;

//calculo timestamp de las dos fechas
$timestamp1 = mktime(0,0,0,$mes1,$dia1,$ano1);
$timestamp2 = mktime(0,0,0,$mes2,$dia2,$ano2);
```

Luego, podríamos restar los timestamp y convertir los segundos en días:

```
//resto a una fecha la otra
$segundos_diferencia = $timestamp1 - $timestamp2;
//echo $segundos_diferencia;

//convierto segundos en días
$dias_diferencia = $segundos_diferencia / (60 * 60 * 24);
```

Para convertir los segundos en días, como se ha podido observar en el código, hay que dividir entre el número de segundos de un día. (60 segundos de un minuto, por los 60 minutos de una hora, por las 24 horas de un día).
Ahora bien, con un código como el anterior, el valor de los días de diferencia puede tener decimales y ser negativo. Nosotros queremos un número de días entero y positivo. Entonces todavía tendremos que hacer un par de operaciones matemáticas. Primero quitar el signo negativo y luego quitar los decimales.

```
//obtengo el valor absoulto de los días (quito el posible signo negativo)
$dias_diferencia = abs($dias_diferencia);

//quito los decimales a los días de diferencia
$dias_diferencia = floor($dias_diferencia);
```

Los decimales los quitamos simplemente redondeando hacia abajo. Puesto que si tenemos un número decimal de días no ha llegado a un día completo y no nos interesa contabilizarlo.

- **Calcular días entre fechas con SQL:**

SELECT TIMESTAMPDIFF(DAY, '1997-06-21', '2018-05-07') AS dias_transcurridos;

Ejemplo aplicado en caja_backup.
SELECT CODIGO,
TIMESTAMPDIFF(**DAY**,FECHA_ALTA,FECHA_SALIDA)
AS DIAS_EN_ALMACEN **FROM** CAJA_BACKUP **ORDER BY**
dias_en_almacen **desc**;

Modelo Vista Controlador (MVC).

Modelo-Vista-Controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

De manera genérica, los componentes de MVC se podrían definir como sigue:

El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.

La Vista: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.

La cuarta capa.

Normalmente este modelo viene asociado a una cuarta capa DAO (Data Access Object) que será la encargada de hacer de puente entre los controladores y los datos almacenados en las tablas.
Consiste en una clase que se corresponde con otra del modelo pero que solo tiene métodos. Estos métodos se encargan de recoger datos en sus parámetros y dentro de estos métodos se implementa la llamada SQL que se solicita en función de los mismos.

Cuando el requerimiento es de obtención de datos de tablas se encargarán, a partir de lo recibido del SELECT, de montar los objetos para devolverlos a la capa controladora. En sentido contrario, cuando por ejemplo se trate de una inserción, estos métodos recibirán un objeto que dispone de los datos a insertar en la tabla correspondiente. En este caso el método obtendrá los datos de las propiedades y realizará el INSERT oportuno.
Esta capa tiene dos posibles implementaciones, una clase general con todos los métodos de acceso a tablas de nuestro proyecto o diseñar una clase por cada clase de la capa modelo.
Esquema de operación para una clase general: