Entorno tecnológico (Lenguajes)

Florentino Fernández [1] and Roberto Casado-Vara [2]

[1] University of Vigo, Circunvalación ao Campus Universitario, 36310 Vigo, Pontevedra, Spain
verola@uvigo.es
[2] University of Salamanca, Plaza de los Caídos s/n – 37002 – Salamanca, Spain
rober@usal.es

Resumen. Los lenguajes de programación son lo que nos permite programar a las maquinas para que realicen las tareas que necesitamos. Dentro de los lenguajes de programación están los de alto nivel (los mas cercanos al lenguaje humano) y los de bajo nivel (los mas cercanos al lenguaje maquina). Además, estos lenguajes pueden estar enfocados a las aplicaciones, a las paginas web o a las aplicaciones de los movieres. En este capitulo se presenta PHP un lenguaje de alto nivel desarrollado específicamente para realizar paginas web. Por último, se completa el capitulo con un caso de uso de la interacción del lenguaje PHP con una base de datos.

Palabras clave: PHP; Paginas web; MySQL.

Abstract. Programming languages are what allow us to program machines to perform the tasks we need. Among the programming languages are high level (the closest to human language) and low level (the closest to machine language). In addition, these languages can be focused on applications, web pages or mobile applications. In this chapter PHP is presented a high level language developed specifically to make web pages. Finally, the chapter is completed with a case of the use of PHP language interaction with a database.

Keywords: PHP; Websites; MySQL

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

1 Introducción

1.1 ¿Qué es PHP?

PHP es un lenguaje de scripts del lado del servidor que permite definir scripts o secuencias que se ejecutan en el servidor cuando se produce una petición de la entidad PHP. Se incrusta en las páginas HTML estáticas como un lenguaje de programación de estilo clásico, es decir, un lenguaje de programación con variables, sentencias condicionales, bucles, funciones, etc. No es un lenguaje de marcas como podría ser HTML, XML o WML, sino que está más cercano a C o a Javascript.
Al ejecutarse en el servidor no es necesario que el navegador lo soporte, pero el servidor donde están alojadas las páginas PHP debe dar soporte a esta tecnología.

1.2 Historia

PHP nació en otoño de 1994 de manos de Rasmus Lerdorf que, para mantener su página personal, creó un conjunto de binarios CGI escritos en lenguaje C que sustituían los scripts en Perl que usaba anteriormente. Estos binarios recibieron el nombre de *Personal Home Page Tools* (PHP Tools). A las funcionalidades iniciales le añadiría un intérprete de formularios para crear PHP/FI, que incluso incluía soporte para el acceso a bases de datos.
En Junio de 1995 Lerdorf publica PHP en su versión 2 con el fin de mejorar el código y de acelerar la detección de errores. Esta versión ya incluía las funcionalidades básicas que PHP tiene hoy en día, tales como variables estilo Perl, gestión de formularios o la capacidad incrustar código en HTM [1-5]L.
En 1997 Zeev Suraski y Andi Gutmans reescribieron el analizador sintáctico de PHP, lo que sería la base para crear la versión 3. La publicación oficial no llegaría hasta Junio de 1998. Con esta versión también llegaría un cambio del nombre, tomando el nombre actual de *PHP: Hypertext Preprocessor*.
Tras reescribir el analizador sintáctico, Suraski y Gutmans empezaron a trabajar en el núcleo de PHP, creando el Zend Engine en 1999. También fundaron la empresa Zend Technologies en Ramat Gan, Israel. El Zend Engine sería la base de la versión 4 de PHP, publicada en Mayo de 2000. Aunque en Junio de 2004 se publicó la versión 5 con el nuevo Zend Engine II, el soporte de la versión 4 se mantuvo hasta Agosto de 2008 debido a su popularidad. La versión 5 destaca por el soporte mejorado de la programación orientada a objetos y por las mejoras en el rendimiento.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

Actualmente la versión 5 es la única soportada, aunque la versión 6 ya está en desarrollo. En la versión actual, la 5.3, aparte de introducirse diversas mejoras, se han marcado como obsoletas aquellas funciones que serán eliminadas en la versión 6.

1.3 Ventajas

- Es un lenguaje multiplataforma, soportado en sistemas UNIX, Window 32 bits, QNX, Mac, OS/2, etc.
- Soporta casi todos los motores de base de datos actuales, aunque destacan MySQL y PostgreSQL.
- Extensibilidad de sus funcionalidades mediante el uso de módulos (llamados exts o extensiones).
- Existe una amplia comunidad de desarrolladores que emplean PHP y que ayudan en su desarrollo, testeo y documentación.
- Buena documentación en su página oficial (www.php.net) y a través de la amplia comunidad de desarrolladores.
- Es software abierto, con todas las ventajas que esto supone. Especialmente respecto a los costes de licencia.
- La biblioteca nativa de funciones incluida por defecto es realmente amplia. Además existen proyectos tales como PEAR (PHP Extension and Application Repository) y PECL (PHP Extension Community Library) con muchas más bibliotecas que extienden las funcionalidades básicas.
- El desarrollo de aplicaciones suele ser mucho más rápido que con otras plataformas tales como Java, C, C++, etc.

1.4 Desventajas

- Las aplicaciones en PHP suelen tener problemas de escalabilidad. Aunque se está trabajando en ello, es difícil que PHP alcance el nivel de escalabilidad disponible plataformas tales como los servidores de aplicaciones.
- A pesar de que puede funcionar en distintas plataformas, PHP tiene como entorno principal a la combinación de Linux, Apache y MySQL. Esto hace que bajo otros entornos su gestión y configuración no sea siempre sencilla.
- La migración de un servicio desarrollado sobre PHP a otra plataforma es complicada, teniendo en cuenta que se trata de un lenguaje embebido en la capa de presentación (HTML), no estándar.
- PHP carece, por el momento, de interfaz gráfica de gestión, herramientas de monitorización o de características visuales para un sencillo mantenimiento.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

2 Sintaxis del Lenguaje

2.1 Incrustación de PHP en HTML

Existen diversas formas de introducir código PHP en un documento HTML. Las más habituales son:

```
1. <? echo 'Primer método de delimitar código PHP'; ?>
<?= 'salida rapida' ?>Esto es una abreviatura de"<?echo 'salida rapida'?>"
2. <?php echo 'Segundo método, el más usado'; ?>
3. <script language="php">
echo 'Algunos editores (como el FrontPage), sólo entienden este método';
</script>
4. <% echo 'Método de compatibilidad con ASP'; %>
```

Aunque sólo la 2ª y 3ª están disponibles siempre, los otros dos modos se pueden activar/desactivar al compilar PHP.
Como en la mayoría de los lenguajes de programación, un final de línea se marca en PHP con un ; (punto y coma). Cuando una instrucción está seguida por una etiqueta de cierre de PHP (?>), se puede ignorar el punto y coma, pues esta etiqueta lo incluirá automáticamente si no está presente [6-10].
En PHP hay tres formas de introducir comentarios en el código:

```
<?
   /* Este es un comentario al estilo C
     que ocupa varias líneas
   */
   // Este es un comentario al estilo de C++ que sólo ocupa una línea
   # Este comentario también acaba al final de la línea
?>
```

2.2 Variables, Tipos y Constantes

2.2.1 Variables

El nombre de una variable es una cadena alfanumérica que identifica una zona de memoria donde se almacena su valor. Existen ciertas limitaciones en cuanto al nombre que se puede asignar a una variable:

- Se compone por una secuencia de caracteres que pueden ser una letra perteneciente al conjunto de caracteres ASCII Extendido, un dígito o el carácter de subrayado (_).
- La variable debe comenzar con el símbolo dólar ($), al que seguirá una cadena de caracteres que comenzará con una letra o el carácter de subrayado.
- PHP diferencia entre mayúsculas y minúsculas.

El conjunto de los nombres válidos en PHP puede definirse con la siguiente expresión regular: [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*
El nombre $this es un nombre de variable especial y está reservado.
Las variables en PHP no necesitan ser definidas antes de su utilización y pueden contener cualquier valor. La asignación de un valor a una variable se realiza con el operador de asignación =.
PHP permite el uso de *variables de variables*. Para entenderlo veamos un ejemplo donde se define una variable denominada *$coche* con el valor “SEAT”.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

```
$coche = "SEAT";
```

si a continuación se realiza:

```
$$coche = "IBIZA";
```

se asigna el valor “IBIZA” a la variable cuyo nombre es el contenido en la variable *$coche*. Es decir, la asignación anterior es equivalente a:

```
$SEAT = "IBIZA";
```

2.2.2 Constantes

Como en otros muchos lenguajes de programación, PHP permite la definición de constantes, que permiten utilizar identificadores que no cambian de valor durante la ejecución de un programa. El siguiente ejemplo define la constante PI con el valor real 3.141593.

```
define(‘PI’, 3.141593);
```

Como vemos, las constantes se definen mediante la función “define”. El primer parámetro es una cadena de texto con el nombre de la constante. El segundo valor es el que se le asignará a la constante. Las constantes siguen las mismas reglas de nomenclatura que las variables, salvo que no deben empezar por $. Por convención los identificadores están en mayúsculas.
Una vez definida una constante estará accesible desde toda la aplicación usando su nombre (sin comillas) [11-17].
Existen un conjunto de constantes predefinidas llamadas “constantes mágicas” (*magic constants*). El nombre de estas variables esta sufijado y prefijado por dos caracteres de subrayado (__). El valor de estas constantes depende del contexto en el que se usen. Algunos ejemplos de este tipo de variables se hallan en la Tabla 1.

Nombre	Descripción
__LINE__	Número de la línea actual.
__FILE__	Ruta completa y nombre del fichero actual.
__DIR__	Ruta completa del directorio en el que se encuentra el fichero actual.
__FUNCTION__	Nombre de la función actual.
__CLASS__	Nombre de la clase actual.
__METHOD__	Nombre del método actual.
__NAMESPACE__	Nombre del espacio de nombres actual.

Tabla 1: Ejemplo de constantes mágicas en PHP.

2.2.3 Tipos

PHP no es un lenguaje fuertemente tipado, puesto que las variables no necesitan definirse antes de su uso y, además, se puede asignar diferentes tipos de valor a la misma variable durante la ejecución de un programa, transformándose el tipo interno de la variable de forma transparente. Aún así, PHP permite realizar la conversión de un tipo a otro utilizando *casting* explícito.
En PHP existen cuatro tipos de datos básicos, dos tipos complejos y dos tipos especiales. Todos ellos se muestran en la Tabla 2.

Tipos Básicos	
Tipo	**Descripción**
integer	Número entero, positivo o negativo
float (double)	Número con notación decimal o científica.
boolean	Valor de tipo booleano (*true* o *false*).
string	Cadena de caracteres.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

Tipos Complejos	
Tipo	**Descripción**
array	Array asociativo. Actúa como un mapa en el que se asocian valores a claves.
object	Las variables pueden contener instancias de objetos de una determinada clase
Tipos Especiales	
Tipo	**Descripción**
resource	Las variables de tipo resource hacen referencia a un recurso externo, tal como puede ser un flujo de datos, un fichero o una base de datos.
NULL	Las variables sin valor son de este tipo.

Tabla 2: Tipos de datos en PHP.

Dado que PHP no es un lenguaje fuertemente tipado, hay ocasiones en las que puede ser necesario conocer el tipo de datos que se almacena en una determinada variable. Para ello, se pueden utilizar las funciones que se muestran en la Tabla 3 (no se listan todas), que permiten determinar si una variable almacena o no un valor de un tipo de dato concreto.

Función	Descripción
is_int(var)	Devuelve true si var es de tipo entero.
is_long(var)	Devuelve true si var es de tipo entero.
is_double(var)	Devuelve true si var es de tipo real.
is_float(var)	Devuelve true si var es de tipo real.
is_real(var)	Devuelve true si var es de tipo real.
is_string(var)	Devuelve true si var es una cadena.
is_numeric(var)	Devuelve true si var es de tipo numérico.
is_array(var)	Devuelve true si var es un array.
is_object(var)	Devuelve true si var es un objeto.

Tabla 3: Funciones de comprobación de tipos.

La conversión de tipos en PHP se puede llevar a cabo por medio de funciones o por medio de "casts". Los "casts" funcionan de modo similar a como lo hacen en C: el nombre del tipo deseado se escribe entre paréntesis antes de la variable que se va a convertir. La Tabla 4 muestra los tipos de "cast" admitidos.

Cast	Convierte a
(int), (integer)	integer
(bool), (boolean)	boolean
(float), (double), (real)	float
(string)	string
(array)	array
(object)	object
(unset)	NULL

Tabla 4: Tabla de conversión de tipos con "casts".

Algunas de las funciones que permiten convertir el tipo de las variables son intval(), floatval(), doubleval() o strval(). En el siguiente ejemplo se puede ver cómo actúan.

```
<!-- conversion.php -->
<HTML>
<BODY>
<?php
        $cadena = "1234";
        $numero = intval($cadena);
```

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

```
        echo "El valor numérico es " . $numero . "<BR/>";
        $cadena = "1010";
        // Se convierte de binario a decimal
        $numero = intval($cadena, 2);
        echo "El valor numérico es " . $numero . "<BR/>";
        $cadena = "97.76";
        $real = doubleval($cadena);
        echo "El valor numérico es " . $real . "<BR/>";
        $numero = 1234;
        $cadena = strval($numero);
        echo "El valor de cadena es " . $cadena . "<BR/>";
?>
</BODY>
</HTML>
```

2.3 Operadores

En esta sección se listan los principales operadores existentes en PHP con una breve descripción de su funcionamiento. En el apartado final se muestra la precedencia de los operadores.

2.3.1 Operadores Generales y Especiales

Operador	Nombre	Ejemplo	Descripción
=	Asignación	$a = 5	Asigna un valor a una variable.
=	Asignación combinada	$a += 5	Ejecuta una operación binaria empleando la $a como un operador. El resultado es asignado a $a. Se puede emplear con todos los operadores aritméticos binarios, con el operador de unión de arrays y con los operadores de strings.
@	Control de errores	@file('no existe')	Evita que se muestren mensajes de error al ejecutar una operación.
``	Ejecución	`ls –l`	El contenido se intenta ejecutar como un comando de shell.
.	Concatenación de string	'Hola'.' Mundo'	Concatena dos cadenas de texto.
? :	Ternario	$a ? $b : $c	Si $a verdadero entonces devuelve $b. Si no devuelve $c. $b y $c también pueden ser código ejecutable.
instanceof	Comprobación de tipo	$a instanceof MiClase	TRUE si $a es un objeto de la clase MiClase.

2.3.2 Operadores Aritméticos

Operador	Nombre	Ejemplo	Descripción
-	Negación	-$a	Convierte un valor en su opuesto.
+	Suma	5 + 6	Suma dos números.
-	Resta	5 - 3	Resta dos números.
*	Multiplicación	3 * 5	Multiplica dos números.
/	División	4 / 3	Divide dos números.
%	Módulo	15 % 4	Devuelve el resto de dividir dos números. En el ejemplo el resultado es 3.
++	Pre-incremento	++$a	Incrementa $a una unidad, después devuelve el $a.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

++	Post-incremento	$a++	Devuelve un $a, después incrementa $a en una unidad.
--	Pre-decrecimiento	--$a	Decrece $a una unidad, después devuelve el $a.
--	Post-decrecimiento	$a--	Devuelve un $a, después decrece $a en una unidad.

2.3.3 Operadores a Nivel de Bit

Operador	Nombre	Ejemplo	Descripción
&	Y	$a & $b	Bits que están a activos en $a y en $b se ponen activos. El resto inactivos.
|	O	$a | $b	Bits que están a activos en $a o en $b se ponen activos. El resto inactivos.
^	O exclusivo	$a ^ $b	Bits que están a activos en $a o en $b, pero no en ambos, se ponen activos. El resto inactivos
~	No	~$a	Se invierten los bits.
<<	Desplazamiento a la izquierda	$a << $b	Los bits de $a se mueven $b pasos a la izquierda.
>>	Desplazamiento a la derecha	$a >> $b	Los bits de $a se mueven $b pasos a la derecha.

2.3.4 Operadores Lógicos

Operador	Nombre	Ejemplo	Descripción
and	Y	$a and $b	TRUE si $a y $b son TRUE.
&&	Y	$a && $b	TRUE si $a y $b son TRUE.
or	O	$a or $b	TRUE si $a o $b son TRUE.
||	O	$a || &b	TRUE si $a o $b son TRUE.
xor	O exclusivo	$a xor $b	TRUE si $a o $b son TRUE, pero no ambos.
!	No	!$a	TRUE si $a no es TRUE.

2.3.5 Operadores de Comparación

Operador	Nombre	Ejemplo	Descripción
==	Igual	$a == $b	TRUE si $a igual a $b.
===	Idéntico	$a === $b	TRUE si $a igual a $b y, además, son del mismo tipo.
!=	Distinto	$a != $b	TRUE si $a distinto de $b.
<>	Distinto	$a <> $b	TRUE si $a distinto de $b.
!==	No idéntico	$a !== $b	TRUE si $a distinto de $b o no son del mismo tipo.
<	Menor que	$a < $b	TRUE si $a menor que $b.
>	Mayor que	$a > $b	TRUE si $a mayor que $b.
<=	Menor o igual que	$a <= $b	TRUE si $a menor o igual que $b.
>=	Mayor o igual que	$a >= $b	TRUE si $a mayor o igual que $b.

2.3.6 Operadores de Arrays

Operador	Nombre	Ejemplo	Descripción
+	Unión	$a + $b	Unión de $a y $b.
==	Igual	$a == $b	TRUE si $a y $b tienen las mismas parejas clave/valor.
===	Idéntico	$a === $b	TRUE si $a y $b tienen las mismas parejas clave/valor, en el mismo orden y con los mismos tipos.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

!=	Distinto	$a != $b	TRUE si $a no es igual que $b.
<>	Distinto	$a <> $b	TRUE si $a no es igual que $b.
!==	No idéntico	$a !== $b	TRUE si $a no es idéntico a $b.

2.3.7 Precedencia de Operadores

Asociatividad	Operadores	Información Adicional
No asociativo	clone new	Objetos
Izquierda	[	Array
No asociativo	++ --	Aritméticos
No asociativo	~ - (int) (float) (string) (array) (object) (bool) @	Tipos
No asociativo	instanceof	Tipos
Derecha	!	Lógico
Izquierda	* / %	Aritmético
Izquierda	+ - .	Aritmético y de string
Izquierda	<<>>	A nivel de bit
No asociativo	<<= >>= <>	Comparación
No asociativo	== != === !==	Comparación
Izquierda	&	A nivel de bit
Izquierda	^	A nivel de bit
Izquierda	|	A nivel de bit
Izquierda	&&	Lógico
Izquierda	||	Lógico
Izquierda	? :	Ternario
Derecha	= += -= *= /= .= %= &= |= ^= <<= >>=	Asignación
Izquierda	and	Lógico
Izquierda	xor	Lógico
Izquierda	or	Lógico
Izquierda	,	Muchos usos

Los paréntesis pueden ser usados para forzar y dar más precedencia a las expresiones que contienen.

2.4 Estructuras de Control

Cualquier script PHP está compuesto por un conjunto de sentencias. Una sentencia puede ser una asignación, una llamada a una función, un bucle, una sentencia condicional o incluso una sentencia que no hace nada (sentencia vacía). Las sentencias suelen acabar con un punto y coma. Además, las sentencias también pueden ser agrupadas en grupos de sentencias mediante la encapsulación con llaves. Un grupo de sentencias es, a su vez, una sentencia. A continuación se describen los distintos tipos de sentencias existentes [18-25].

2.4.1 Sentencias Condicionales

Las sentencias condicionales permiten ejecutar o no ciertas instrucciones dependiendo del resultado de evaluar una condición. Existen dos tipos: **if** y **switch**.

2.4.1.1 Sentencia if/elseif/else

La sentencia *if* permite ejecutar un bloque de código sólo si se cumple una determinada condición. La sentencia *if* puede ir acompañada de una sentencia *else*, que hará que se ejecute un bloque de código si la condición del *if* no se cumple. También se puede encontrar la sentencia *elseif* acompañándola, la cual actúa como un bloque *if* si la anterior sentencia *if* no se cumple.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

El siguiente ejemplo muestra la sintaxis completa de la sentencia *if*.

```
if (condición)
{
        instrucciones_si_true
}
elseif (otra_condición)          // Opcional. Puede haber varios.
{
        instrucciones_si_true
}
else                                    // Opcional
{
        instrucciones_si_false
}
```

A continuación vemos un ejemplo de uso de *if*.

```
<!-- if.php -->
<HTML>
<HEAD>
<TITLE>Ejemplo de if con PHP</TITLE>
</HEAD>
<BODY>
<?php
        $a = 8;
        $b = 3;
        if ($a < $b)
        {
                echo "a es menor que b";
        }
        else
        {
                echo "a no es menor que b";
        }
?>
</BODY>
</HTML>
```

Las sentencias *if* poseen una sintaxis abreviada que puede ser usada en lugar de la sintaxis habitual con el fin de mejorar la legibilidad del código.

```
if..else..endif

if (condición) :
instrucciones_si_true
else :
        instrucciones_si_false
endif
```

2.4.1.2 Sentencia switch

El otro tipo de sentencia condicional es la sentencia *switch*. Está formada por una variable de entrada y varios bloques de código. Dependiendo del valor de la variable de entrada se ejecutará un determinado bloque de código. Vemos un ejemplo a continuación.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

```
<!-- switch.php -->
<HTML>
<HEAD>
<TITLE>Ejemplo de switch con PHP</TITLE>
</HEAD>
<BODY>
<?php
        $posicion = "arriba";
        switch($posicion)
        {
                case "arriba":
                        echo "La variable contiene";
                        echo " el valor arriba";
                        break;
                case "abajo":
                        echo "La variable contiene";
                        echo " el valor abajo";
                        break;
                default:
                        echo "La variable contiene otro valor";
                        echo " distinto de arriba y abajo";
        }
?>
</BODY>
</HTML>
```

La sentencia *switch* también tiene una sintaxis abreviada.

```
<?php
switch ($i):
        case 0:
                echo "i igual a 0";
                break;
        case 1:
                echo "i igual a 1";
                break;
        case 2:
                echo "i igual 2";
                break;
        default:
                echo "i no es igual a 0, 1 o 2";
endswitch;
?>
```

2.4.2 Bucles

Los bucles permiten ejecutar un bloque de código repetidas veces. El número de veces que se ejecuta el código de los bucles se controla, normalmente, con una condición.

2.4.2.1 Sentencia while

La sentencia del bucle while se ejecuta mientras se cumpla una determinada condición. La condición es comprobada antes de cada ejecución del bucle. Su sintaxis es la siguiente.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
while (condición)
        sentencia
```

O en su versión abreviada.

```
while (condición):
sentencia
        ...
endwhile;
```

2.4.2.2 Sentencia do/while

La sentencia *do/while* es muy similar a la sentencia *while*, con la diferencia de que la condición de ejecución se comprueba tras cada ejecución del bucle.

```
do{
        instrucciones;
} while (condición)
```

2.4.2.3 Sentencia for

Este tipo de bucles es el más complejo. La sentencia *for* también se ejecuta mientras se cumpla una condición, pero además de esta condición el bucle *for* contiene una expresión de inicialización y una de actualización. La expresión de inicialización se ejecuta antes de comenzar el bucle. La expresión de actualización se ejecuta en cada iteración del bucle, antes de comprobar la condición.

```
for (inicialización; condición; actualización)
{
        instrucciones;
}
```

Su sintaxis abreviada es la siguiente.

```
for (inicialización; condición; actualización):
        instrucciones;
        ...
endfor;
```

2.4.2.4 Sentencia foreach

El bucle *foreach* permite recorrer un array elemento a elemento. Los elementos del array se almacenan en una variable que estará accesible dentro del bucle. Posee dos sintaxis, una de las cuales permite acceder al valor de la clave de cada elemento.

```
foreach (array as $valor)                   // Sintaxis 1
        sentencia
foreach (array as $clave => $valor)         // Sintaxis 2
        sentencia
```

2.4.2.5 Sentencias break y continue

Estas dos sentencias son sentencias especiales que permiten variar la ejecución normal de los bucles. La sentencia *break* hace que se detenga la ejecución de un bucle, saltando la ejecución a la sentencia que se encuentre a continuación del bucle. La sentencia *continue* hace que se detenga una iteración del bucle y se ejecute de inmediato la comprobación de la condición del bucle (o se pase al siguiente elemento en el caso de los bucles *foreach*).

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

Inclusión de Código desde Archivos

PHP permite ejecutar código PHP existente en otros archivos. Para ello se realiza la inclusión del código existente en los archivos externos. Existen dos sentencias (que actúan como funciones) que permiten realizar esta inclusión: *require*() e *include*().
Tanto la función *require*() como *include*() incluyen y evalúan el fichero indicado como parámetro; permite insertar y ejecutar un archivo en el programa que se esté cargando. Es adecuado para la inclusión de constantes o funciones que no se van a modificar, además de cualquier código de presentación que se considera repetitivo (encabezados y pies de página, menús, etc.), por ejemplo:

```
require("constantes.php");
include("funciones.php");
```

require()e *include*() son idénticas, excepto en el comportamiento que presentan ante un error al no encontrar el fichero indiciado: *include*() produce un *Warning*(E_WARNING) y *require*() produce un *Error Fatal* (E_ERROR).
Es preciso destacar que se debe tener cuidado para no incluir dos o más veces un mismo fichero si en él se definen funciones, ya que se presentaría un error de redefinición de una misma función. Para evitar este problema están las sentencias *include_once*() y *require_once*(), que sólo incluyen el fichero una vez durante la evaluación de todo el *script*. La desventaja de estas dos funciones es que resultan más costosas de ejecutar, al necesitar comprobar si ya se ha incluido el fichero.

2.5 Funciones

Cuando se diseña una aplicación con funciones, lo mejor es situar la definición de las funciones al comienzo, de tal manera que estén definidas antes de su uso. Esto facilita la lectura del código. El formato general de definición de una función en PHP es el siguiente:

```
function nombreFuncion($parametro1, $parametro2, ...)
{
        // Cuerpo de la función
}
```

Los argumentos que se pasan a una función pueden ser variables, números u objetos. Una función también puede devolver un resultado utilizando la sentencia *return*.
Existen dos formas distintas de realizar el paso de parámetros a una función:

- **Paso de parámetros por valor.** Éste es el método por defecto utilizado en PHP. Al parámetro de la función se le asigna una copia del valor con el que es invocado. La función no puede modificar el valor de la variable.
- **Paso de parámetros por referencia.** El parámetro contiene una referencia a la variable con la que fue invocado. La función puede modificar el valor de la variable y los cambios que se produzcan serán visibles fuera de la función. Para indicar en PHP el paso de una variable por referencia se utiliza el carácter & en la declaración que se haga de la misma en la función.

```
<!-- parametros.php -->
<HTML>
<BODY>
<?php
        function intercambiar (&$var1, &$var2) {
```


ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
                $aux = $var1;
                $var1 = $var2;
                $var2 = $aux;
        }
        $n1 = 7;
        $n2 = 16;
        echo "Antes de la llamada a la función: " .$n1. ", " .$n2. "<BR/>";
        intercambiar($n1, $n2);
        echo “Después de la llamada a la función: " .$n1. ", " .$n2. "<BR/>";
?>
</BODY>
</HTML>
```

Los parámetros de las funciones pueden tener un valor por defecto. Esto funciona de modo similar a como ocurre en C++. Si al invocar una función no se asigna un valor al parámetro tomará su valor por defecto. Es importante que los parámetros con un valor por defecto vayan después de los parámetros sin valor por defecto.

```
<?php
function hacerCafe($tipo = "cappuccino")
{
        return "Haciendo una copa de $tipo.<BR/>";
}
echo hacerCafe();               // Haciendo una copa de cappuccino.
echo hacerCafe(null);           // Haciendo una copa de .
echo hacerCafe("espresso");     // Haciendo una copa de espresso.
?>
```

Las funciones pueden devolver un valor haciendo uso de la sentencia *return*. Se puede devolver cualquier valor, incluyendo arrays u objetos.

```
<?php
function cuadrado($num)
{
        return $num * $num;
}
echo cuadrado (4);   // imprime '16'.
?>
```

2.6 Ámbito y Tipo de Variables

Una variable *local* es aquella que se define dentro de una función. Su ámbito se restringe a la función en la que se define y, por lo tanto, no es visible desde fuera de la misma. Una variable *global* es la que se define fuera de las funciones y, por lo tanto, puede ser accedida desde cualquier función, pero para ello éstas deben estar declaradas en la función utilizando la palabra reservada *global*. Otra forma de acceder a variables globales se realiza mediante el *array* $GLOBALS que almacena todas las variables globales que hayan sido definidas a lo largo del código PHP.
El siguiente ejemplo ilustra el empleo de variables globales en PHP:

```
<!-- globals.php -->
<HTML>
<BODY>
<?php
        function asignarUno ()
```

ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239

```
	{
		global $numero; // $numero es una variable global
		$numero = 1;
	}

	function asignarDos ()
	{
		$GLOBALS["numero"] = 2; // $numero es variable global
	}

	$numero = 7;
	echo "El valor de numero es " .$numero. "<BR/>";
	asignarUno();
	echo "El valor de numero es " .$numero. "<BR/>";
	asignarDos();
	echo "El valor de numero es " .$numero. "<BR/>";
?>
</BODY>
</HTML>
```

PHP también permite la definición de variables *estáticas*. Una variable estática es una variable local a una función cuyo valor persiste entre las distintas invocaciones a la función. El siguiente programa contabiliza el número de veces que se invoca a la función contar.

```
<!-- static.php -->
<HTML>
<BODY>
<?php
	function contar ()
	{
		static $veces = 0;
		$veces++;
		echo "Se ha ejecutado contar " .$veces. " veces <BR/>";
	}
	for ($i = 0; $i < 80; $i++)
		contar();
?>
</BODY>
</HTML>
```

2.7 Manejo de Cadenas

En los lenguajes de script de los servidores Web, resulta muy importante el manejo de cadenas de texto, pues el objetivo principal es generar un documento y servirlo al cliente. Es por ello que PHP ofrece un conjunto muy completo de funciones para la manipulación de *strings* o cadenas de caracteres, además de disponer de una sintaxis especial para las cadenas que facilita su manipulación [26-30].

Cuando se le asigna un valor a una cadena de texto se puede hacer de dos modos, empleando comillas simples ('texto') o empleando comillas dobles ("texto"). La diferencia entre ambos métodos es que con comillas dobles el texto se procesa y con comillas simples no.

El procesado del texto incluye la detección y sustitución de las variables contenidas por su valor. Esto resulta más sencillo de comprender con un ejemplo.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
<!-- strings.php -->
<HTML>
<HEAD>
<TITLE>Ejemplo de strings con PHP</TITLE>
</HEAD>
<BODY>
        <?php
                $suma = 2 + 3;
                echo "La suma de 2 y 3 es: {$suma}";      // La suma de 2 y 3 es: 5
                echo 'La suma de 2 y 3 es: {$suma}';      // La suma de 2 y 3 es: {$suma}
        ?>
</BODY>
</HTML>
```

El uso de llaves no es obligatorio pero evita errores en la delimitación de los nombres de las variables. Las comillas dobles sólo se deberían usar cuando hay que procesar el texto contenido, pues suponen una carga en el tiempo de ejecución.
La Tabla 5 muestra la especificación de las funciones más importantes de manipulación de cadenas de caracteres.

Función	Descripción
echo(string arg1 [,string argn])	Imprime las cadenas pasadas como parámetro.
print(string c)	Imprime la cadena pasada como parámetro.
printf(formato [,valores])	Imprime con formato. Similar a la función de C.
string chr(int valor)	Devuelve el carácter asociado al código ASCII pasado como parámetro.
int ord(string c)	Devuelve el código ASCII asociado al carácter c.
string trim(string c)	Elimina del principio y del final de la cadena c los blancos.
string ltrim(string c)	Elimina del principio de la cadena c los blancos.
string rtrim(string c)	Elimina del final de la cadena c los blancos.
string str_replace(string c1, string c2, string c)	Sustituye en la cadena c todas las ocurrencias de la cadena c2 por c1.
int strlen(string c)	Devuelve la longitud de la cadena.
string strtolower(string c)	Convierte la cadena a minúscula.
string strtoupper(string c)	Convierte la cadena a mayúscula.
string substr(string c, int pos [, int l])	Devuelve un fragmento de la cadena a partir de pos. El parámetro l determina la máxima longitud de la cadena devuelta.
int strops(string c, string s [, int pos])	Localiza en c la ocurrencia de s. El argumento pos (opcional) indica el inicio de la búsqueda.
string strchr(string cad, char c)	Devuelve la subcadena que comienza en la primera aparición del carácter.
int strcmp(string c1, string c2)	Compara dos cadenas de caracteres.

Tabla 5: Principales funciones de manejo de cadenas en PHP.

2.8 Manejo de Arrays

Como ocurre en otros lenguajes de programación, un *array* agrupa bajo un mismo nombre de variable diversos valores. A continuación se describen las particularidades que presentan los *arrays* en PHP.
En realidad en PHP no existen *arrays* como en los lenguajes de programación habituales, sino que son siempre mapas asociativos clave-valor (similar a tablas hash). Las claves pueden ser de tipo cadena o entero (cualquier otro intento de utilizar otro tipo para la clave se convertirá a alguno de estos dos, por ejemplo los flotantes se truncan). No existen por tanto dos tipos de *arrays*, unos con índice entero y otros con índice de cadena. Un mismo *array* puede incluso tener valores asociadas a claves de diferente tipo, unos con clave de tipo entero y otros con clave de tipo cadena. Consecuencia de lo anterior, el siguiente ejemplo se muestra un *array* válido [31-40].

```
$matriz[“clave1”] = “valor1”;                    // clave de tipo cadena
$matriz[“clave2”] = “valor2”;     // clave de tipo cadena
```

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

```
$matriz[10] = “valor3”;                    // clave de tipo entero
```

Otra consecuencia de la naturaleza de mapa de los *arrays es* que pueden tener índices no secuenciales. El siguiente ejemplo define un *array* con tres elementos e índices no secuenciales. Si se desea conocer el número de elementos de un *array* se puede utilizar la función *count()*. Por ejemplo:

```
$vector[40] = 77;
$vector[10] = 45.67;
$vector[1] = "miércoles";
print(count($vector)); // imprime 3
/*Sin embargo, lo anterior no impide crear un array similar a uno clásico, que además se puede recorrer de una forma cásica también:*/
$dias[0] = “lunes”;
$dias[1] = “martes”;
$dias[2] = “miércoles”;
$dias[3] = “jueves”;
$dias[4] = “viernes”;
$dias[5] = “sabado”;
$dias[6] = “domingo”;
for ($i=1; $i<7; $i++){
        echo “Dia $i es: ”.$dias[$i].”<BR/>”;
}
```

Los elementos de un *array* pueden ser de distinto tipo, por ejemplo:

```
$vector[0] = 77;
$vector[1] = 45.67;
$vector[2] = "miércoles";
```

Un *array* también se puede crear con el constructor *array*. Por ejemplo:

```
$alumnos = array("Juan", "Manuel", "Fernando");
$alumno = array("nombre"=>"Fernando", "Apellidos" => "Díaz Gómez");
```

Un aspecto importante a tener en cuenta con los *array*sal ser asociativos, no siempre puede accederse a los elementos del *array* por su posición. Cuando se quiere recorrer de forma secuencial los elementos de un *array* asociativo, se puede utilizar el bucle *foreach*.

```
<!-- foreach.php -->
<HTML>
<BODY>
<?php
        $alumno["Nombre"] = "Fernando";
        $alumno["Apellidos"] = "Díaz Gómez";
        $alumno["Edad"] = 32;
        foreach ($alumno as $elemento)
        {
                echo " $elemento";
                echo “<BR/>”;
        }
?>
</BODY>
</HTML>
```


ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

El bucle *foreach* recorre secuencialmente el *array* $alumno y almacena en cada iteración en la variable $elemento el valor actual.
Otra forma de recorrer secuencialmente los elementos de un *array* asociativo es utilizar la función *each*(). El funcionamiento de esta función se puede apreciar en el siguiente ejemplo:

```
<!-- each.php -->
<HTML>
<BODY>
<?php
        $alumno["Nombre"] = "Fernando";
        $alumno["Apellidos"] = "Díaz Gómez";
        $alumno["Edad"] = 32;
        while ($elemento = each($alumno))
        {
                echo " $elemento[0]: $elemento[1]";
                echo "<BR/>";
        }
?>
</BODY>
</HTML>
```

Estas dos formas de acceso no son adecuadas cuando se quiere retroceder al elemento anterior, o se desea ir al primero o al último elemento. En este caso se pueden utilizar las funciones que se describen la Tabla 6. A todas ellas se les pasa como parámetro el *array* sobre el que se desean aplicar.

Función	Descripción
reset()	Coloca el puntero de posición en el primer elemento del *array*.
end()	Coloca el puntero de posición en el último elemento del *array*.
next()	Avanza al siguiente elemento. Cuando llega al final devuelve *false*.
prev()	Retrocede al elemento anterior. Cuando llega al principio devuelve *false*.
current()	Devuelve el contenido del elemento actual. No desplaza el puntero. Devuelve *false* cuando no apunta a un elemento.

Tabla 6: Funciones para el recorrido de *arrays* en PHP

2.9 Clases y Objetos

Con la versión 5 de PHP se introdujo un soporte amplio para clases y objetos. La versión 4 ya soportaba clases, pero era un soporte muy limitado.
Los objetos en PHP son un tipo especial de dato, en lo que respecta a su tratamiento. Si una variable contiene un método, en realidad no contiene el objeto en sí, si no que almacena una referencia al objeto. Por lo tanto, son tratados con el mismo tratamiento que las referencias.

2.9.1 Conceptos Básicos

class

La definición de una clase comienza con la palabra clave *class* seguida por el nombre de la clase. A continuación se define el cuerpo de la clase, encerrado entre llaves. El nombre de la clase sigue las mismas normas que las variables.
Una clase puede contener constantes, variables (llamadas propiedades) y funciones (llamadas métodos).

```
<?php
class MiClase {
```

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

```
        // declaración de una constante
        const PI = 3.14;
        // declaración de una propiedad
        public $var = 'propiedad';
        // declaración de un método
        public function muestraVar() {
                echo $this->var;
        }
}
?>
```

El ejemplo anterior muestra clase simple. En él se puede ver una declaración de una constante, de una propiedad y de un método. También se puede observar el uso de la pseudo-variable **$this**. Esta pseudo-variable representa al objeto mismo que la contiene.
En PHP el operador de acceso a propiedades y a métodos es una flecha (->).

new

La creación de objetos se realiza mediante la palabra clave new. Para crear un objeto de una clase se debe usar new seguido del nombre de la clase que se desea instanciar. En el caso de que la clase reciba parámetros en su constructor (ver apartado Constructores y Destructores), deberán ir a continuación entre paréntesis.

```
<?php
        $instancia = new MiClase();
?>
```

extends

PHP5 admite herencia de clases por medio de la palabra reservada extends. Cuando una clase (subclase) hereda de otra (superclase), los métodos y propiedades de la superclase pasan a la subclase. Tan sólo se admite herencia simple, es decir, cada clase puede heredar, como máximo, de una clase.
Los métodos y propiedades pueden ser sobreescritos redeclarándolos en la subclase con el mismo nombre que tienen en la superclase. Para evitar esto se puede usar la palabra reservada **final** en una clase para que sus subclases no realizar la sobreescritura. Se puede acceder a los métodos o propiedades sobreescritos usando **parent::**.

```
<?php
        class SubClase extends MiClase{
                // Redefine un método de la clase padre.
                function muestraVar() {
                        echo "Extendiendo una clase\n";
                        parent::muestraVar();
                }
        }
        $extended = new SubClase();
        $extended->muestraVar();
?>
```

2.9.2 Propiedades

Las variables que son miembros de clases se denominan “propiedades”, “campos” o “atributos”. Las propiedades se definen acompañadas de un operador de visibilidad (ver apartado Visibilidad) y puede asignárseles un valor inicial en la misma declaración. Otro modo de asignar un valor a las propiedades es mediante el uso de un constructor (var apartado Constructores y Destructores).

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

Cuando una propiedad se define como estática (operador *static*) se dice que es una "propiedad de clase".
Se puede acceder a las propiedades de los objetos mediante el operador de acceso (->). En el caso de estar en el contexto de una clase, el acceso se puede hacer con $this.
El acceso a las propiedades de clase y a las constantes se hace con el operador "::". Igual que la pseudo-propiedad $this da acceso a un objeto dentro de sí mismo, existe la palabra reservada **self** que hace referencia a la misma clase.

```
<?php
        class Propiedades {
                public $var = 'Hola';
                public static $varStatic = 'Adios';
                public getVar() {
                        return $this->var;
                }
                public getVarStatic() {
                        return self::$varStatic;
                }
        }
        $objeto = new Propiedades();
        // Ambos mostrarían el contenido de var
        echo $objeto->var;
        echo $objeto->getVar();
        // Ambos mostrarían el contenido de varStatic
        echo $objeto->getVarStatic();
        echo $objeto::$varStatic;
?>
```

2.9.3 Constantes

Es posible definir constantes en las clases. Las constantes actúan como variables que nunca cambian de valor, además su nombre no comienza con $. El valor asignado a las constantes debe ser una expresión constante y no se aceptan asignaciones tales como variables, propiedades, resultados de operaciones matemáticas, llamadas a funciones, etc.
El acceso a las constantes se hace exactamente igual que como se accede a las variables estáticas, con el operador "::".

2.9.4 Visibilidad

La visibilidad de una propiedad o método se puede definir prefijando su declaración con las palabras clave:

- **public**: Los miembros de este tipo pueden ser accedidos desde cualquier contexto.
- **protected**: Los miembros de este tipo sólo pueden ser accedidos por la clase que los declara y por sus subclases.
- **private**: Los miembros de este tipo sólo pueden ser accedidos por la clase que los declara.

El ejemplo que viene a continuación muestra cómo funciona la visibilidad en el caso de las propiedades. El acceso a los métodos funciona exactamente igual.

```
<?php
        class MiClase {
                public $public = 'Public';
```

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

```
                protected $protected = 'Protected';
                private $private = 'Private';
                function imprime() {
                        echo $this->public;
                        echo $this->protected;
                        echo $this->private;
                }
        }
        $obj = new MiClase();
        echo $obj->public; // Funciona
        echo $obj->protected; // Error Fatal
        echo $obj->private; // Error Fatal
        $obj->imprime(); // Funcionan todos los echo
        // MiClase2 es una subclase de MiClase
        class MiClase2 extends MiClase {
                function imprime() {
                        echo $this->public;
                        echo $this->protected;
                        echo $this->private;
                }
        }
        $obj2 = new MiClase2();
        echo $obj2->public;   // Funciona.
        echo $obj2->private;  // No está definido.
        echo $obj2->protected;// Error Fatal.
        $obj2->imprime();     // Funcionan los echo Public y Protected.
                                        // Private no está definido.
?>
```

2.9.5 Constructores y Destructores

Existen dos métodos especiales en las clases de PHP5. Su signatura es la siguiente:

- void **__construct** ([mixed $args [, $...]])
- void **__destruct** (void)

Si está presente el método __construct será invocado cuando un objeto de la clase sea creado. Este método se denomina "constructor" y permite inicializar el estado de un objeto.
Si está presente el método __destruct será invocado en el momento en el que el objeto sea explícitamente eliminado, cuando se eliminen todas sus referencias o cuando se finalice la ejecución. Este método se denomina "destructor" y generalmente se emplea para liberar recursos que pudiese tener bloqueados el objeto

2.9.6 Interfaces

Una interfaz es una tipo de clase en la que sólo se declara el esqueleto de la misma. El contenido de una interfaz es únicamente la declaración de sus métodos, pero sin el cuerpo de los mismos. También es posible declarar constantes en una interfaz.
No se pueden crear instancias de una interfaz, pues un objeto siempre debe tener una implementación completa de sus métodos.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

Las interfaces se declaran con la palabra clave **interface** en lugar de class. La implementación de una interfaz se hace de modo similar a la herencia, salvo que en este caso se usa la palabra reservada **implements** en lugar de extends.
A diferencia de la herencia, una clase puede implementar varias interfaces, separando por comas el nombre de las interfaces que implementa. La herencia y la implementación de interfaces son compatibles. También es posible extender una interfaz con otra.

```
<?php

        // Declaración de la interfaz.
        interface iPlantilla {
                public function setVariable($name, $var);
                public function getVariable($name);
        }
        // Implementación de la interfaz.
        class Plantilla implements iPlantilla{
                private $vars = array();
                public function setVariable($name, $var){
                        $this->vars[$name] = $var;
                }
                public function getVariable($name) {
                        return $this->vars[$name];
                }
        }
?>
```

2.9.7 Clases Abstractas

Las clases abstractas están a medio camino entre una interfaz y una clase normal. Permiten la declaración de métodos con cuerpo y métodos sin cuerpo. Tanto la clase como los métodos sin cuerpo deben ser declarados como abstractos, mediante el uso del modificador **abstract**.
Como en el caso de las interfaces, no se puede crear una instancia de una clase que sea abstracta, pues un objeto debe tener la implementación completa de sus métodos.
La herencia de una clase abstracta funciona igual que con las clases normales, pero la subclase debe implementar todos los métodos abstractos o ser declarada también como abstracta.

```
<?php
        abstract class ClaseAbstracta {
                // Este método debe ser implementado en las subclases.
                abstract protected function getValor();
                // Método normal.
                public function imprimir() {
                        print $this->getValor() . "\n";
                }
        }
        class ClaseConcreta extends ClaseAbstracta {
                protected function getValor() {
                        return "ClaseConcreta";
                }
        }
?>
```

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

2.9.8 Clonación de Objetos

Como se ha comentado, los objetos son tratados como referencias. Esto hace que los objetos no puedan ser copiados directamente, pues si igualamos una variable a otra que contenga un objeto lo que se estará copiando será la referencia, no el objeto en sí.
Para poder realizar una copia de un objeto se debe "clonar", para lo que se usa la palabra reservada **clone**. Cuando se clona un objeto se crear un nuevo objeto de la misma clase con una copia de las propiedades del objeto original. En el caso de que una propiedad contenga una referencia, la misma propiedad en el objeto clonado también contendrá la misma referencia.
Cuando se crea una copia de un objeto no se llama a su constructor, pues se supone que el objeto ya ha sido inicializado en su original. Aún así existe un método que permite realizar modificaciones en un objeto justo después de haber sido clonado. Este método se llama **__clone()** y es invocado justo después de que la copia se cree y se copien las propiedades.
En el ejemplo siguiente vemos un ejemplo de clonación y de uso del método __clone().

```
<?php
        class Clonable {
                static $instancias = 0;
                public $instancia;
                public function __construct() {
                        $this->instancia = ++self::$instancias;
                }
                public function __clone() {
                        $this->instancia = ++self::$instancias;
                }
        }
        $original = new Clonable();
        $clon = clone $original;
?>
```

3 Variables PHP

PHP proporciona un gran número de variables predefinidas que facilitan enormemente la programación de aplicaciones web. Son *arrays* asociativos que dan acceso a cada detalle específico de cada objeto cuyo nombre representa: detalles del servidor, control de sesión, formularios, manejo de *cookies*, etc.
Para ver un listado muy completo de las variables disponibles en PHP, así como detalles de configuración del servidor y del módulo de PHP, puede crearse un fichero PHP que únicamente contenga la siguiente línea:

```
<? phpinfo() ?>
```

Al ejecutarlo desde la ventana del navegador web se mostrarán las variables PHP disponibles.

3.1 Variables Superglobales

A partir de la versión 4.1.0 de PHP, existen ciertas variables globales predefinidas que se denominan *Superglobales*. Se trata de variables que son globales de forma automática, esto es, no es preciso incluir la declaración *global $VARIABLE* para que sean accesibles desde un *script*. La Tabla 7 muestra las variables superglobales disponibles a partir de PHP 4.1.0 y su nombre en versiones anteriores de PHP.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

Superglobal	Nomenclatura antigua
$_SERVER	$HTTP_SERVER_VARS
$_ENV	$HTTP_ENV_VARS
$_SESSION	$HTTP_SESSION_VARS
$_GET	$HTTP_GET_VARS
$_POST	$HTTP_POST_VARS
$_COOKIE	$HTTP_COOKIE_VARS
$_FILES	$HTTP_POST_FILES
$_REQUEST	No hay equivalente

Tabla 7: Variables superglobales y sus equivalentes en nomenclatura antigua.

3.1.1 $_SERVER

Esta variable permite que los *scripts* accedan a un conjunto muy extenso de información que proporciona el servidor web sobre el que está ejecutando el *script*. Esta información atañe a los diversos aspectos de la comunicación entre el cliente y el propio *script* que ejecuta el servidor.

3.1.2 $_ENV

Todo *script* PHP hereda del entorno en el que se está ejecutando una serie de variables de entorno. Cuáles sean estas y cuál sea su significado dependerá del entorno en que se ejecute el *script*. Se debe tener en cuenta que un *script* no debe nunca fiarse de estas variables, dado que no son controladas por el servidor web y, por lo tanto, pueden tener cualquier valor.

3.1.3 $_SESSION

Desde PHP, y también desde otros lenguajes como ASP o JSP, se dispone de un mecanismo denominado "sesión" que permite la gestión de variables que mantiene su valor durante toda la visita o secuencia de accesos de un cliente al servidor. Mediante esta técnica, una secuencia de accesos por parte de un cliente a un *script* PHP (o a un conjunto de ellos) podrá ser tratada como un todo.
Para gestionar sesiones, PHP proporciona una serie de funciones prefijadas por "session_", entre las que se encuentran:

- *session_start*(). Todo *script* PHP que maneje sesiones debe comenzar con una llamada a esta función que verifica**y carga**los datos de la sesión si ya existe y, si no existe, crea una nueva. Esta llamada producirse siempre antes de que el *script*PHP produzca alguna salida (por ejemplo,con *echo*).
- *session_destroy*(). Cuando es invocada, termina la sesión con el usuario actual, aunque no vacía el array *$_SESSION* para el resto del script actual.

El siguiente *script* implementa un juego que consiste en adivinar un número secreto producido de forma aleatoria entre 0 y 9999. Para llevar la cuenta del número generado, así como para controlar el número de intentos realizado, se utiliza una sesión en la cual se registran dos variables denominadas *secreto* e *intentos*.

```
<?php
    session_start();
?>
<!-- juego.php -->
<HTML>
<HEAD>
<TITLE>Adivina</TITLE>
</HEAD>
```

ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
<BODY>
<?php
	$numero = (int) $_REQUEST['numero'];
	if (empty($secreto) || $_REQUEST['reinicio'] == "Reiniciar") {
		echo "Intente adivinar el número secreto.<BR/><BR/>";
		$_SESSION['secreto'] = rand(0, 9999);
		$_SESSION['intentos'] = 0;
	} else if (empty($numero)) {
		echo "Intente adivinar el número secreto.<BR/><BR/>";
	} else if ($_SESSION['secreto'] < $numero) {
		$_SESSION['intentos']++;
		echo "El número secreto es menor que " . $numero . "<BR/>";
		echo "Lleva usted " . $_SESSION['intentos'] . " intentos!<BR/>";
	} else if ($_SESSION['secreto'] > $numero) {
		$_SESSION['intentos']++;
		echo "El número secreto es mayor que " . $numero . "<BR/>";
		echo "Lleva usted ". $_SESSION['intentos'] . " intentos!<BR/>";
	} else {
		echo "<BR/>";
		echo "ENHORABUENA, el número secreto era " . $numero . "!<BR/>";
		echo "<BR/>";
		session_destroy();
		exit();
	}
?>
<FORM method="POST">
	Introduzca un número entero entre 0 y 9999<BR/>
	Número: <INPUT name="numero" type="text"></INPUT>
	<INPUT name="envio" type="submit" value="Probar"/>
	<INPUT name="reinicio" type="submit" value="Reiniciar"/>
</FORM>
</BODY>
</HTML>
```

El siguiente *script* muestra cómo se puede crear una sesión que mantiene un nombre de usuario.

```
<?php
	session_start(); //Siempre se debe poner por si ya hay sesion

	// Se comprueba si se han introducido los datos de login
	if (isset($_POST['login'])) {
		$_SESSION['login'] = $_POST['login'];
		header("Location:login.php"); //limpiar el post redirigiendo

	// Se comprueba si se quiere salir
	} else if (isset($_GET['salir'])) {
		// Destrucción de una sesion
		$_SESSION = array();
		session_destroy();
		header("Location:login.php"); //limpiar el get redirigiendo
	}
	// No se ha pulsado ni salir ni entrar
```


ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
        else {
?>
                <HTML>
                <HEAD>MiniLogin</HEAD>
                <BODY>
                <H1>MiniLogin</H1>
                <!-- Un enlace a otro sitio donde se ve lo que hay en sesion -->
                <A href="usasesion.php">[Ir a otro sitio]</A><BR/><BR/>
                <?php
                // Se puede estar ya identificados
                if (isset($_SESSION['login'])) {
                        echo "hola ".$_SESSION['login'];
                ?>
                <BR/>
                <A href="?salir=si">Salir</A>
<?php
                } else {
                // Si no se ha identificado lo solicitamos con formulario
?>
                        <FORM method="post" acion="login.php">
                                Nombre: <INPUT name="login" type="text"><BR/>
                                <INPUT type="submit" value="Entrar"/>
                        </FORM>
<?php
                }
?>
                </BODY>
                </HTML>
<?php
        }
?>
```

El siguiente *script* complementa al anterior con una página donde se visualiza el nombre del usuario, si éste se ha identificado, demostrando cómo un valor guardado en sesión se puede recuperar en cualquier otra página del sitio que se esté implementando.

```
<?php
        session_start();
?>
<!-- usasesion.php -->
<HTML>
<HEAD>Usa sesión</HEAD>
<BODY>
        <H1>Usa sesión</H1>
        <?php
        if (isset($_SESSION['login'])) {
                echo "Hola ".$_SESSION['login']."<BR/>";
        } else {
                echo "No te has identificado<BR/>";
        }
        ?>
        <A href="login.php">Volver</A>
```

ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
</BODY>
</HTML>
```

3.1.4 $_GET

Es un *array* asociativo que contiene toda la información que se le pasó al *script* mediante el método de invocación GET.
El siguiente ejemplo implementa un esquema básico de un programa de búsqueda que utiliza el método GET para recoger un patrón a buscar.

```
<!-- get.php -->
<HTML>
<HEAD>
<TITLE>Búsqueda</TITLE>
</HEAD>
<BODY>
<?php
        $patron = $_GET['patron'];
        if (empty($patron)) {
?>
<FORM method="GET">
        Introduzca el patrón de búsqueda:<BR/>
        <INPUT name="patron" type="text"/>
        <INPUT name="submit" value="Buscar"/>
</FORM>
<?php
        } else {
                // En este punto se realizaría la búsqueda...
                echo "Resultados de la búsqueda del patrón: $patron<BR/>...";
                // A continuación se presentarían los resultados...
        }
?>
</BODY>
</HTML>
```

3.1.5 $_POST

Es un *array* asociativo que contiene toda la información que se le pasó al *script* mediante el método de invocación POST.
Usando el método POST en vez del método GET, no existe limitación a la cantidad de información que un formulario puede suministrar y, por otro lado, tal información no aparecerá visible en la URL.
El siguiente *script* utiliza el método POST para enviar un texto extenso de opinión sobre un asunto.

```
<!-- post.php -->
<HTML>
<HEAD>
<TITLE>Opinión</TITLE>
</HEAD>
<BODY>
<?php
        $texto = $_POST['texto'];
```


ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
        if (empty($texto)) {
?>
        <FORM method="POST">
                <H2>Nos interesa mucho su opinión!</H2>
                <TEXTAREA name="texto" rows="10" cols="30"></TEXTAREA>
                <INPUT type="submit" value="Enviar">
        </FORM>
<?
        } else {
                // En este punto $texto se guardaría adecuadamente...
                echo "<H2>Todas las opiniones y sugerencias son bienvenidas.</H2>";
                echo "<H2>Gracias!.</H2>";
                echo "Usted opina:<HR/><PRE>$texto</PRE><HR/>";
        }
?>
</BODY>
</HTML>
```

3.1.6 $_COOKIE

En PHP, el *array* asociativo $_COOKIE contiene la información relativa a cada *cookie* que el *script* recibe del navegador.
Por otro lado, cuando desde un *script* se quiera establecer el valor de una nueva *cookie* o cambiar el de una existente, se deberá hacer uso de la función de php *setcookie*, que se comenta brevemente a continuación:

```
setcookie(nombre, valor, caducidad, camino, dominio, segura)
```

El protocolo HTTP impone la restricción de que si se desea enviar una *cookie* al navegador, se deberá hacer antes de que el *script* produzca ninguna salida. Es por esta razón que el código PHP de manejo de las cookies suele aparecer al principio de los *scripts*.
Esta función recibe un número variable de argumentos. Todos menos el primero (*nombre*) son opcionales y todos menos *caducidad* son cadenas de caracteres.
Si sólo aparece el primer argumento o se utiliza una fecha de caducidad del pasado, se borrará la *cookie* en el navegador [41-43].
Caducidad. Es la fecha de caducidad indicada como un número entero de segundos desde medianoche del 1 de enero de 1970.
El siguiente *script* crea y utiliza una *cookie* como contador del número de veces que se ha accedido al *script*. Si se accede repetidas veces al *script*, puede observarse cómo el contador va incrementándose, pero si se deja de acceder por más de 60 segundos, la *cookie* se perderá, con lo que posteriores accesos volverán a contar desde 1.

```
<?php
        $contador = $_COOKIE['contador'];
        $contador++;
        setcookie("contador", $contador, time()+60);
?>
<!-- cookie.php -->
<HTML>
<HEAD>
<TITLE>Contador</TITLE>
</HEAD>
<BODY>
```

ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239

```
        <H1><?php echo $contador; ?></H1>
</BODY>
</HTML>
```

3.1.7 $_FILES

Mediante el método POST es posible incluso mandar archivos completos del cliente al servidor. Para manejar este tipo de información, PHP proporciona la variable $_FILES, que es un *array* asociativo que contiene toda la información necesaria (nombre, tamaño, tipo, etc.) para cada archivo enviado.
El siguiente *script* permite que el usuario seleccione un archivo de su máquina local para enviarlo al servidor y poder manejarlo en el propio *script*.

```
<!-- file.php -->
<HTML>
<HEAD>
<TITLE>Entrega</TITLE>
</HEAD>
<BODY>
<?php
        $fichero = $_FILES['fichero'];
        if (empty($fichero)) {
?>
        <FORM enctype="multipart/form-data" method="POST">
                <H2>Escoja el archivo que desea enviar:</H2>
                <INPUT name="fichero" type="file"/><BR/>
                <INPUT type="submit" value="Enviar"/>
        </FORM>
<?php
        } else {
                echo "Los datos relativos al archivo suministrado son:<HR/>";
                echo "Nombre original " . $fichero['name'] . "<HR/>";
                echo "Tipo de archivo " . $fichero['type'] . "<HR/>";
                echo "Tamaño del fichero " . $fichero['size'] . "<HR/>";
                echo "Nombre temporal " . $fichero['tmp_name'] . "<HR/>";
                if (!empty($fichero['error']))
                        echo "Error ocurrido " . $fichero['error'] . "<HR/>";
        }
?>
</BODY>
</HTML>
```

Un ejemplo más avanzado es el que se muestra en el siguiente *script* que implementa algo similar a un servidor FTP. Se muestra un formulario para subir ficheros y a la vez se listan los ficheros ya subidos, pudiendo consultarlos o eliminarlos.

```
<!-- miniftp.php -->
<HTML>
<HEAD>
<TITLE>MiniFTP</TITLE>
</HEAD>
<BODY>
```


ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
<H1>MiniFTP</H1>
<?php
        // Recepcion de un fichero
        $fichero = $_FILES['fichero'];
        if (!empty($fichero)) {
                if (empty($fichero['error'])){
                        move_uploaded_file($fichero['tmp_name'], "./upload/".$fichero['name']);
                        echo "Fichero subido correctamente";

                        // Botón de volver
?>
<INPUT type="button" onClick="javascript:document.location.href='miniftp.php'" value="Volver"></INPUT>
<?php
                } else { // Ha habido un error
                        echo "Error ocurrido " . $fichero['error'] . "<HR/>";
                }

        // Borrado de un fichero
        } else if(isset($_GET['borrar'])) {
                // Seguridad: Evitamos que alguien envíe un fichero y trate de
                // ascender directorios
                if (strstr($_GET['borrar'],"/")) {
                        //Esto es un ataque
                        die("Sistema atacado, el administrador fue avisado");
                }
                if (is_file("./upload/".$_GET['borrar'])) {
                        if (unlink("./upload/".$_GET['borrar'])) {
                                echo "Fichero borrado correctamente:";
                                echo $_GET['borrar'];
                                echo "<BR/>";
?>
<INPUT type="button" onClick="javascript:document.location.href='miniftp.php'"
value="Volver"></INPUT>
<?php
                        } else {
                                echo "Error al borrar el fichero.<BR/>";
?>
<INPUT type="button" onClick="javascript:document.location.href='miniftp.php'"
value="Volver"></INPUT>
<?php
                        }
                } else {
                        echo "Se ha tratado de borrar un fichero inexistente";
                }

    // Si no se envía ni se borra, se listan los ficheros y se muestra
    // un formulario
    } else {
                //Listado de los ficheros
                if ($gestor = opendir('./upload')) {
```

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239

```
                                        echo "<H2>Ficheros en el servidor:</H2><BR/>";
                                        while (false !== ($archivo = readdir($gestor))) {
                                                // No queremos visualizar directorios ni . o ..
                                                if (!is_file("./upload/".$archivo)) continue;
                                                echo "<A href='./upload/".$archivo."'>";
                                                echo $archivo."</A>";

                                                // Botón de borrar
        ?>
        <INPUT type="button" onClick="javascript:document.location.href='?borrar=<?=$archivo?>'"
value="Borrar" >
        <BR/>
<?php
                                        }
                                        closedir($gestor);
                                }
        ?>
        <!-- Formulario de envio -->
        <FORM enctype="multipart/form-data" method="POST">
                <H2>Escoja el archivo que desea enviar:</H2>
                <INPUT name="fichero" type="file"><BR/>
                <INPUT type="submit" value="Enviar">
        </FORM>
        <?php
                }
        ?>
</BODY>
</HTML>
```

3.1.8 $_REQUEST

Este *array* asociativo contiene la concatenación de la información presente en las variables superglobales $_GET, $_POST, $_COOKIE.
Esto significa que, si al programar un *script* no importa cuál es el origen de la variable a la que se quiere acceder, se puede optar por referirse a ella a través de este *array*.
El siguiente ejemplo muestra varias formas de referirse a una variable que fue suministrada a través del método POST.

```
<?php
        echo $variable;                 // Sólo si register_globals es true, no recomendado
        echo $HTTP_POST_VARS['variable'];       // Obsoleto, en desuso
        echo $_POST['variable'];                // Recomendada
        echo $_REQUEST['variable'];             // Otra posibilidad
?>
```

4 Gestión de Bases de Datos desde PHP

Uno de los puntos fuertes de PHP siempre ha sido el soporte de base de datos. Las aplicaciones Web suelen hacer uso de bases de datos para almacenar información, por lo que un buen lenguaje de scripts para servidores debe proporcionar soporte para trabajar con ellas.

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

En PHP existen dos modos de trabajo con bases de datos, a través de bibliotecas de funciones específicas para cada tipo de base de datos o empleando una capa de abstracción que permita al programador abstraerse del sistema gestor de base de datos subyacente.
Actualmente, PHP proporciona soporte para tres tipos de capas de abstracción:

- **DBA (Database Abstraction Layer)**: Proporciona soporte para el trabajo con bases de datos del estilo de Berkeley DB. Este tipo de bases de datos se caracterizan porque su gestión se realiza directamente desde una biblioteca y no es necesario instalar un sistema gestor de base de datos.
- **ODBC (Open DataBase Connectivity)**: Es un API estándar para el uso de sistemas gestores de bases de datos. Este estándar nació, precisamente, para conseguir que los lenguajes de programación fuesen independientes de las bases de datos.
- **PDO (PHP Data Objects)**: Es una extensión que define una interfaz ligera y consistente para el acceso a bases de datos. Esta interfaz es común para todas las bases de datos, con lo que se consigue la abstracción del tipo de base de datos empleado. Los objetos de PDO encapsulan los drivers de las distintas bases de datos.

Las bibliotecas de funciones específicas de cada base de datos se caracterizan por que sus funciones están prefijadas con un nombre asociado con la base de datos a la que dan acceso. Así, por ejemplo, las funciones de Informix comienzan por ifx_ y las de SQLite por sqlite_.
En los repositorios PEAR y PECL existen bibliotecas adicionales para el trabajo con bases de datos.

4.1 Ejemplo de Trabajo con MySQL

Vamos a ver un como es el trabajo con bases de datos en PHP, ejemplificado con la biblioteca específica de MySQL. Las funciones de MySQL se identifican por el prefijo mysql_. Existen alrededor de 50 funciones, pero para hacernos una idea de cómo funciona tan sólo vamos a emplear las funciones de la Tabla 8.

Función	Descripción
descriptor mysql_connect(string serv, string user, string passwd)	Abre una conexión MySQL con el servidor.
bool mysql_close(descriptor myd)	Cierra una conexión con el servidor MySQL.
bool mysql_select_db(string bdatos, descriptor myd)	Selecciona una base de datos almacenada en el servidor.
bool mysql_query(string qry, descriptor myd)	Envía una *query* al gestor MySQL.
objeto mysql_fetch_object(objeto reslt)	Recupera un objeto del resultado de una *query*.
bool mysql_free_result(objeto reslt)	Libera los recursos asignados a una consulta.

Tabla 8: Resumen de las funciones de PHP para acceso a MySQL.

Lo primero que hay que hacer para trabajar con una base de datos en PHP es establecer una conexión, a través de la cual nos comunicaremos con la base de datos. El siguiente ejemplo muestra como se establece una conexión y se realiza una consulta a una base de datos.

```
<!—- conexion.php -->
<?php
        $desc = mysql_connect("localhost", "root", "");
        if (!$desc)
                die("No se puede conectar con el servidor");
        mysql_select_db("libros", $desc);
        $resultado = mysql_query("SELECT * FROM libros", $desc);
        while ($obj = mysql_fetch_object($resultado))
                echo "ISBN: $obj->isbn AUTOR: $obj->autor <BR/>";
```

ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239

```
        mysql_free_result($resultado);
        mysql_close($desc);
?>
```

A continuación, para ilustrar el uso de las funciones relacionadas con la gestión de una base de datos MySQL se exponen dos ejemplos: una consulta a una base de datos que almacena los datos relacionados con libros y una actualización. Los campos contenidos en la base de datos son: *ISBN*, *Autor*, *Título*, *Editorial* y *Año de publicación*. El formulario que se muestra a continuación permite que el usuario rellene un campo para buscar todos los libros que encajen en el criterio de búsqueda.

```
<!-- buscar.htm -->
<HTML>
<BODY>
        <FORM METHOD="GET" NAME="formulario" ACTION="conssq.php">
                <B>ISBN  </B><INPUT name="FISBN"><BR/>
                <B>AUTOR  </B><INPUT name="FAUTOR"><BR/>
                <B>TITULO  </B><INPUT name="FTITULO"><BR/>
                <B>EDITORIAL  </B><INPUT name="FEDITORIAL"><BR/>
                <B>AÑO  </B><INPUT name="FANO"><BR/>
                <INPUT type="submit" value="Enviar">  
                <INPUT type="reset" value="Borrar">
        </FORM>
</BODY>
</HTML>
```

La aplicación *conssq.php* resuelve la consulta y genera una tabla con los libros de la base de datos que siguen el criterio especificado por el usuario. El *script* PHP se conecta al servidor MySQL y selecciona la base de datos *libros*. Se toman como entrada las variables FISBN, FAUTOR, FTITULO, FEDITORIAL y FANO que son las que ha introducido el usuario en el formulario anterior. La función *construye_query* genera la *query* SQL con los campos que ha rellenado el usuario, que no están vacíos. Por ejemplo, si el usuario introduce el valor *rodri* en el campo *autor* y *web* en el campo *titulo*, se genera la *query*:

```
select * from libros where autor like "%rodri%" and titulo like "%web%";
```

Esta *query* se pasa como parámetro a *mysql_query* que realiza la consulta a la base de datos. Cada uno de los registros resultado de la consulta se obtiene llamando a *mysql_fetch_object*, que devuelve un objeto con todos los campos de la base de datos que se imprimen como una línea en una tabla. En el campo ISBN se introduce un enlace al *scriptzoom.php*, que generará un nuevo formulario para su actualización. El listado de *conssq.php* se adjunta a continuación:

```
<!-- conssq.php -->
<HTML>
<BODY BGCOLOR=”#ffffff”>
<H1>Consulta de libros en una base de datos</H1>
<?php
        function construye_query($isbn, $autor, $titulo, $editorial, $ano) {
                if ($isbn != "") {
                        $res_qry = "isbn =" .$isbn;
                } else {
                        if ($autor != "")
                                $res_qry = 'autor like "%'.$autor.'%"';
                        if ($titulo != "") {
```


ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

```
                                        if (strlen($res_qry) > 0)
                                                $res_qry .=" and ";
                                        $res_qry .="titulo like \"%$titulo%\"";
                                }
                                if ($editorial != "") {
                                        if (strlen($res_qry) > 0)
                                                $res_qry .=" and ";
                                        $res_qry .="editorial like \"%$editorial%\"";
                                }
                                if ($ano != "") {
                                        if (strlen($res_qry) > 0)
                                                $res_qry .=" and ";
                                        $res_qry .="ano like \"%$ano%\"";
                                }
                        }
                        return "select * from libros where $res_qry;";
                }

                $conexion = mysql_connect(“localhost”, “root”, “”);
                if (!$conexion)
                        die(“No se puede conectar a la base de datos”);

                $bd = mysql_select_db("libros", $conexion);
                $query = con-
struye_query($_GET['FISBN'],$_GET['FAUTOR'],$_GET['FTITULO'],$_GET['FEDITORIAL'],$_GET['FANO']);

                $resultado = mysql_query($query, $conexion);
                echo "QUERY: " . $query;
                if ($resultado) {
                        echo "<TABLE BORDER=1><TR><TD>ISBN</TD><TD>AUTOR</TD><TD>TITULO</TD></TR>";

                        while ($entrada = mysql_fetch_object($resultado)) {
                                echo "<TR><TD><A HREF='zoom.php?FISBN=$entrada->isbn'>$entrada->isbn</A></TD>";
                                echo "<TD> $entrada->autor</TD>";
                                echo "<TD> $entrada->titulo</TD>";
                        }
                        echo "</TABLE>";
                        mysql_free_result($resultado);
                } else {
                        echo "No se ha encontrado ningún registro<BR/>";
                }
                mysql_close($conexion);
?>
</BODY>
</HTML>
```

La aplicación *zoom.php* muestra la información completa del libro de la base de datos cuyo ISBN se ha pasado como parámetro. Este programa realiza una consulta con la *query*

```
select * from libros where isbn = $_GET['FISBN'];
```

ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239

y muestra el resultado de la misma en un formulario. Este formulario permite al usuario modificar algunos de sus campos y realizar una actualización llamando al script *update.php*. El listado del *script zoom.php* se incluye a continuación:

```
<!-- zoom.php -->
<HTML>
<HEAD>
<TITLE>Resultado de consulta</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffff">
<H1>Consulta de libros en una base de datos</H1>
<?php
	$conexion = mysql_connect("localhost", "root", "");
	if (!$conexion)
		die("No se puede conectar a la base de datos");

	$bd = mysql_select_db("libros", $conexion);
	$query = "select * from libros where isbn = ".$_GET['FISBN'].";";
	$resultado = mysql_query($query, $conexion);
	if ($resultado) {
		echo "<FORM method=GET action=\"update.php?FISBN=$entrada->isbn\">";
		echo "<TABLE>";
		while ($entrada = mysql_fetch_object($resultado)) {
			echo "<TR><TD>ISBN</TD><TD><INPUT ";
			echo "NAME=FISBN TYPE=TEXT SIZE=10 VALUE='$entrada->isbn'/></TD></TR>";
			echo "<TR><TD>AUTOR</TD><TD><INPUT ";
			echo "NAME=FAUTOR TYPE=TEXT SIZE=30 VALUE='$entrada->autor'/></TD></TR>";
			echo "<TR><TD>TITULO</TD><TD><INPUT ";
			echo "NAME=FTITULO TYPE=TEXT SIZE=40 VALUE='$entrada->titulo'/></TD></TR>";
			echo "<TR><TD>EDITORIAL</TD><TD><INPUT ";
			echo "NAME=FEDITORIAL TYPE=TEXT SIZE=30 VALUE='$entrada-
>editorial'/></TD></TR>";
			echo "<TR><TD>AÑO DE EDICIÓN</TD><TD><INPUT ";
			echo "NAME=FANO TYPE=TEXT SIZE=8 VALUE='$entrada->ano'/></TD></TR>";
		}
		echo "</TABLE>";
		echo '<INPUT TYPE="submit" VALUE="Enviar"/>  ';
		echo '<INPUT TYPE="reset" VALUE="Borrar"/>';
		echo "</FORM>";
		mysql_free_result($resultado);
	} else {
		echo "No se ha encontrado ningún registro<BR/>";
	}
	mysql_close($conexion);
?>
</BODY>
</HTML>
```

El último *script* que se muestra en este ejemplo es *update.php* que realiza la actualización de los campos que se pasan en las variables FISBN, FAUTOR, FEDITORIAL y FANO que ha rellenado

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

el usuario mediante el *script zoom.php*. En este caso se conecta al servidor de base de datos y se selecciona la base de datos *libros*. A continuación, se construye la *query*:

```
update libros set autor = $_GET['FAUTOR'], titulo = $_GET['FTITULO'], editorial = $_GET['FEDITORIAL'], ano =
$_GET['FANO'] where isbn = $_GET['FISBN'];
```

que actualiza el registro cuyo ISBN coincide con el campo ISBN que ha pasado el usuario. Esta *query* actualiza la tabla *libros* con el contenido de las variables citadas anteriormente. El código de *update.php* se muestra a continuación:

```
<!-- update.php -->
<HTML>
<HEAD>
<TITLE>Actualización de un registro</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffff">
<H1>Consulta de libros en una base de datos</H1>
<?php
        $conexion = mysql_connect("localhost", "root", "");
        if (!$conexion)
                die("No se puede conectar a la base de datos");
        $bd = mysql_select_db("libros", $conexion);
        $query = "update libros set autor = \"".$_GET['FAUTOR']."\", titulo = \"".$_GET['FTITULO']."\",
        $editorial = \"".$_GET['FEDITORIAL']."\", ano = \"".$_GET['FANO']."\" where isbn = \"".$_GET['FISBN']."\";";
echo $query;
        $resultado = mysql_query($query, $conexion);
        if ($resultado)
                echo "Actualización realizada correctamente para ISBN ".$_GET['FISBN'];
        else
                echo "Se ha producido un error al actualizar el ISBN ".$_GET['FISBN'];

        mysql_close($conexion);
?>
</BODY>
</HTML>
```

5 Bibliografía

PHP Home Page
http://www.php.net
PEAR - PHP Extension and Application Repository
http://pear.php.net/
PECL :: The PHP Extension Community Library
http://pecl.php.net/
PHP – Wikipedia
http://en.wikipedia.org/wiki/PHP

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

References

1. Adam Macintosh, Ming Feisiyau, Mohammed Ghavami (2014). Impact of the Mobility Models, Route and Link connectivity on the performance of Position based routing protocols. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
2. Ana Silva, Tiago Oliveira, José Neves, Paulo Novais (2016). Treating Colon Cancer Survivability Prediction as a Classification Problem. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1
3. Anna Vilaro, Pilar Orero (2013). User-centric cognitive assessment. Evaluation of attention in special working centres: from paper to Kinect. DCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
4. Casado-Vara, R., & Corchado, J. (2019). Distributed e-health wide-world accounting ledger via blockchain. Journal of Intelligent & Fuzzy Systems, 36(3), 2381-2386.
5. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
6. Casado-Vara, R., de la Prieta, F., Prieto, J., & Corchado, J. M. (2018, November). Blockchain framework for IoT data quality via edge computing. In Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems (pp. 19-24). ACM.
7. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
8. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
9. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
10. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
11. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
12. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
13. Chamoso, P., Rivas, A., Martín-Limorti, J. J., & Rodríguez, S. (2018). A Hash Based Image Matching Algorithm for Social Networks. In Advances in Intelligent Systems and Computing (Vol. 619, pp. 183–190). https://doi.org/10.1007/978-3-319-61578-3_18
14. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
15. Choon, Y. W., Mohamad, M. S., Deris, S., Illias, R. M., Chong, C. K., Chai, L. E., … Corchado, J. M. (2014). Differential bees flux balance analysis with OptKnock for in silico microbial strains optimization. PLoS ONE, 9(7). https://doi.org/10.1371/journal.pone.0102744
16. Corchado, J. M., & Fyfe, C. (1999). Unsupervised neural method for temperature forecasting. Artificial Intelligence in Engineering, 13(4), 351–357. https://doi.org/10.1016/S0954-1810(99)00007-2
17. Corchado, J., Fyfe, C., & Lees, B. (1998). Unsupervised learning for financial forecasting. In Proceedings of the IEEE/IAFE/INFORMS 1998 Conference on Computational Intelligence for Financial Engineering (CIFEr) (Cat. No.98TH8367) (pp. 259–263). https://doi.org/10.1109/CIFER.1998.690316
18. Costa, Â., Novais, P., Corchado, J. M., & Neves, J. (2012). Increased performance and better patient attendance in an hospital with the use of smart agendas. Logic Journal of the IGPL, 20(4), 689–698. https://doi.org/10.1093/jigpal/jzr021
19. Daniel Fuentes, Rosalía Laza, Antonio Pereira (2013). Intelligent Devices in Rural Wireless Networks. DCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
20. Di Mascio, T., Vittorini, P., Gennari, R., Melonio, A., De La Prieta, F., & Alrifai, M. (2012, July). The Learners' User Classes in the TERENCE Adaptive Learning System. In 2012 IEEE 12th International Conference on Advanced Learning Technologies (pp. 572-576). IEEE.
21. Fyfe, C., & Corchado, J. (2002). A comparison of Kernel methods for instantiating case based reasoning systems. Advanced Engineering Informatics, 16(3), 165–178. https://doi.org/10.1016/S1474-0346(02)00008-3
22. Fyfe, C., & Corchado, J. M. (2001). Automating the construction of CBR systems using kernel methods. International Journal of Intelligent Systems, 16(4), 571–586. https://doi.org/10.1002/int.1024

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc

23. García Coria, J. A., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. Expert Systems with Applications, 41(4 PART 1), 1189–1205. https://doi.org/10.1016/j.eswa.2013.08.003
24. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
25. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
26. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
27. Hoon Ko, Kita Bae, Goreti Marreiros, Haengkon Kim, Hyun Yoe, Carlos Ramos (2014). A Study on the Key Management Strategy for Wireless Sensor Networks. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
28. Hugo López-Fernández, Miguel Reboiro-Jato, José A. Pérez Rodríguez, Florentino Fdez-Riverola, Daniel Glez-Peña (2016). The Artificial Intelligence Workbench: a retrospective review. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1
29. Jose-Luis Jiménez-García, David Baselga-Masia, Jose-Luis Poza-Luján, Eduardo Munera, Juan-Luis Posadas-Yagüe, José-Enrique Simó-Ten (2014). Smart device definition and application on embedded system: performance and optimization on a RGBD sensor. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
30. Leonardo Ochoa-Aday, Cristina Cervelló-Pastor, Adriana Fernández-Fernández (2016). Discovering the Network Topology: An Efficient Approach for SDN. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 2
31. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). A particle dyeing approach for track continuity for the SMC-PHD filter. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637583&partnerID=40&md5=709eb4815eaf544ce01a2c21aa749d8f
32. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). Random finite set-based Bayesian filters using magnitude-adaptive target birth intensity. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637788&partnerID=40&md5=bd8602d6146b014266cf07dc35a681e0
33. Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. Applied Mathematics and Computation, 270, 756–767. https://doi.org/10.1016/j.amc.2015.08.059
34. Mar López, Juanita Pedraza, Javier Carbó, José M. Molina (2014). The awareness of Privacy issues in Ambient Intelligence. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
35. Muñoz, M., Rodríguez, M., Rodríguez, M. E., & Rodríguez, S. (2012). Genetic evaluation of the class III dentofacial in rural and urban Spanish population by AI techniques. Advances in Intelligent and Soft Computing (Vol. 151 AISC). https://doi.org/10.1007/978-3-642-28765-7_49
36. Sittón-Candanedo, I., Alonso, R. S., Corchado, J. M., Rodríguez-González, S., & Casado-Vara, R. (2019). A review of edge computing reference architectures and a new global edge proposal. Future Generation Computer Systems, 99, 278-294.
37. Ricardo Faia, Tiago Pinto, Zita Vale (2016). Dynamic Fuzzy Clustering Method for Decision Support in Electricity Markets Negotiation. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 5, n. 1
38. Rodriguez-Fernandez J., Pinto T., Silva F., Praça I., Vale Z., Corchado J.M. (2018) Reputation Computational Model to Support Electricity Market Players Energy Contracts Negotiation. In: Bajo J. et al. (eds) Highlights of Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection. PAAMS 2018. Communications in Computer and Information Science, vol 887. Springer, Cham
39. Rodríguez, S., De La Prieta, F., Tapia, D. I., & Corchado, J. M. (2010). Agents and computer vision for processing stereoscopic images. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6077 LNAI). https://doi.org/10.1007/978-3-642-13803-4_12
40. Rodríguez, S., Tapia, D. I., Sanz, E., Zato, C., De La Prieta, F., & Gil, O. (2010). Cloud computing integrated into service-oriented multi-agent architecture. IFIP Advances in Information and Communication Technology (Vol. 322 AICT). https://doi.org/10.1007/978-3-642-14341-0_29

ISBN: 978-84-9012-859-6

Colección Aquilafuente: AQ-239

41. Sittón, I., & Rodríguez, S. (2017). Pattern Extraction for the Design of Predictive Models in Industry 4.0. In International Conference on Practical Applications of Agents and Multi-Agent Systems (pp. 258–261).
42. Víctor Corcoba Magaña, Mario Muñoz Organero (2014). Reducing stress and fuel consumption providing road information. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
43. Víctor Parra, Vivian López, Mohd Saberi Mohamad (2014). A multiagent system to assist elder people by TV communication. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2

ISBN: 978-84-9012-859-6
Colección Aquilafuente: AQ-239
Ediciones Universidad de Salamanca - cc by nc dc