Programación PHP

Versión

Armando Arce

04 de abril de 2018

Índice general

El objetivo de este sitio es presentar una serie de tutoriales básicos sobre el desarrollo de aplicaciones web utilizando el lenguaje PHP.

Los tutoriales disponibles hasta el momento son los siguientes:

CAPÍTULO 1

Contenidos

1.1 Introducción a PHP

PHP es un lenguaje diseñado para crear contenido HTML. PHP puede ser ejecutado de tres formas: en un servidor web, a través de la línea de comandos, o mediante un cliente GUI.

El lenguaje puede ejecutarse en prácticamente todos los sistemas operativos actuales y en múltiples servidores web. Este también soporta una amplia variedad de bases de datos y cuenta con múltiples librerías para ejecutar procesos comunes.

Una página PHP generalmente consiste de una página HTML con comandos PHP incrustados en ella. El servidor web procesa los comandos PHP y envía la salida al visualizador (browser). Un ejemplo de una página PHP sencilla sería la siguiente:

Una página PHP generalmente consiste de una página HTML con comandos PHP incrustados en ella. El servidor web procesa los comandos PHP y envía la salida al visualizador (browser). Un ejemplo de una página PHP sencilla sería la siguiente:

```
<html>
  <head> <title>Hello, world</title> </head>
  <body>
    <?php echo "Hello, world!"; ?>
  </body>
</html>
```

El comando *echo* de PHP produce la salida que se inserta en la página HTML. Note que el código PHP se escribe dentro de los delimitadores <*?php* y *?*>.

Las instrucciones se separan con *";"*, en el caso de ser la última instrucción no es necesario el punto y coma.

Los comentarios en PHP pueden ser:

- Como en C o C++, /*. . . */ ó //
- Otro tipo de comentario de una línea es #, que comentará la línea en la que aparezca pero sólo hasta el tag *?*> que cierra el código php.

1.1.1 Tipos de Datos

Los tipos de cada variable en PHP no están tan claros como en C. El intérprete asigna el tipo de una variable según el uso que se esté haciendo de ella. Para asignar un tipo fijo a una variable se utiliza la función settype(). Los tipos son:

- Enteros
- Flotantes
- String
- arreglos
- Objetos
- Variables variables

El siguiente ejemplo muestra la utilización de los tipos de datos enteros y flotantes. Los otros tipos de datos se describen más adelante.

```
<html>
<head> <title>Ejemplo 2 </title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php

 #Enteros
 $a = 1234; # número decimal
 $a = -123; # un número negativo
 $a = 0123; # número octal (equivalente al 83 decimal)
 $a = 0x12; /* número hexadecimal (equivalente al 18 decimal) */

 //Flotantes o reales
 $b = 1.234; $b = 1.2e3;

 //Escribimos algo
 print "\n La a= $a y la b= $b <br>\n";
?>

</body>
</html>
```

Hileras de texto

Las hileras de texto pueden estar delimitadas por *» o "*. Si la hilera de texto está delimitada por comillas dobles, cualquier variable incluida dentro de ella será sustituida por su valor (ver y ejecutar el ejemplo anterior). Para especificar el carácter *«* se escapará con el carácter backslash(\ ).

Las operaciones con hileras de texto son exactamente igual que en PERL. Por ejemplo, con strlen se ve el tamaño de una hilera de texto y con el punto (.) se concatenan hileras de texto.

```
<html>
<head> <title>Ejemplo 3 </title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php
 /* Asignando una hilera de texto. */
```

```
 $str = "Esto es una hilera de texto";

 /* Añadiendo a la hilera de texto. */
 $str = $str . " con algo más de texto";

 /* Otra forma de añadir, incluye un carácter de nueva línea  */
 $str .= " Y un carácter de nueva línea al final.\n";
 print "$str <br>\n";

 /* Esta hilera de texto terminará siendo '<p>Número: 9</p>' */
 $num = 9;
 $str = "<p>Número: $num</p>";
 print "$str <br>\n";

 /* Esta será '<p>Número: $num</p>' */
 $num = 9;
 $str = '<p>Número: $num</p>';
 print "$str <br>\n";

 /* Obtener el primer carácter de una hilera de texto  como una vector*/
 $str = 'Esto es una prueba.';
 $first = $str[0];
 print "$str 0->$first <br>\n";

 /* Obtener el último carácter de una hilera de texto. */
 $str = 'Esto es aún una prueba.';
 $last = $str[strlen($str)-1];
 print "$str last->$last <br>\n";
 ?>

</body>
</html>
```

Para hacer conversión de hileras de texto a otros tipos de datos hay que tener en cuenta una hilera de texto se evalúa como un valor numérico, el valor resultante y el tipo se determinan como sigue. La hilera de texto se evaluará como un doble si contiene cualquiera de los caracteres ".", "e", o "E". En caso contrario, se evaluará como un entero. El valor viene dado por la porción inicial de la hilera de texto. Si la hilera de texto comienza con datos de valor numérico, este será el valor usado. En caso contrario, el valor será 0 (cero). Cuando la primera expresión es una hilera de texto, el tipo de la variable dependerá de la segunda expresión.

```
<html>
<head> <title>Ejemplo 4</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php

 $foo = 1 + "10.5";              // $foo es doble (11.5)
 print "$foo <br>\n";
 $foo = 1 + "-1.3e3";            // $foo es doble (-1299)
 print "$foo <br>\n";
 $foo = 1 + "bob-1.3e3";         // $foo es entero (1)
 print "$foo <br>\n";
 $foo = 1 + "bob3";              // $foo es entero (1)
 print "$foo <br>\n";
 $foo = 1 + "10 Cerditos";     // $foo es entero (11)
 print "$foo <br>\n";
```

```
 $foo = 1 + "10 Cerditos"; // $foo es entero (11)
 print "$foo <br>\n";
 $foo = "10.0 cerdos " + 1;        // $foo es entero (11)
 print "$foo <br>\n";
 $foo = "10.0 cerdos " + 1.0;      // $foo es doble (11)
 print "$foo <br>\n";

?>

</body>
</html>
```

Arreglos

Los arreglos en PHP se pueden utilizar tanto como arreglos indexados (vectores) o como arreglos asociativos (tablas hash). Para PHP, no existen ninguna diferencia arreglos indexados unidimensionales y arreglos asociativos. Las funciones que se utilizan para crear arreglos son list() o array() , o se puede asignar el valor de cada elemento del array de manera explícita. En el caso de que no se especifique el índice en un array, el elemento que se asigna se añade al final.

```
<html>
<head> <title>Ejemplo 5</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php

 #forma explícita
 $a[0] = "abc";
 $a[1] = "def";
 $b["foo"] = 13;

 #Añadiendo valores al array
 $a[] = "hola"; // $a[2] == "hola"
 $a[] = "mundo"; // $a[3] == "mundo"

 #mostramos los resultados
 print "a= $a[0] , $a[1] , $a[2] , $a[3] <br>\n";
 print "b[foo]=".$b["foo"]."<br>\n";

?>

</body>
</html>
```

Los arreglos se pueden ordenar usando las funciones asort(), arsort(), ksort(), rsort(), sort(), uasort(), usort(), y uksort() dependiendo del tipo de ordenación que se desee.

Se puede contar el número de elementos de un array usando la función count().

Se puede recorrer un array usando las funciones next() y prev(). Otra forma habitual de recorrer un array es usando la función each().

Los arreglos multidimensionales son bastante simples, para cada dimensión array, se puede añadir otro valor [clave] al final. Los indices de un array multidimensional pueden ser tanto numéricos como asociativos.

```
$a[1]      = $f;          # ejemplos de una sola dimensión
$a["foo"]  = $f;
```

```
$a[1][0]     = $f;         # bidimensional
$a["foo"][2] = $f;         # (se pueden mezclar índices numéricos y asociativos)
$a[3]["bar"] = $f;         # (se pueden mezclar índices numéricos y asociativos)

$a["foo"][4]["bar"][0] = $f;   # tetradimensional!
```

Los arreglos se declarar utilizando la instrucción *array* y se pueden rellenar también usando =>

```
# Ejemplo 1:
$a["color"]     = "rojo";
$a["sabor"]     = "dulce";
$a["forma"]     = "redondeada";
$a["nombre"]    = "manzana";
$a[3]           = 4;

# Ejemplo 2:
$a = array(
     "color" => "rojo",
     "sabor" => "dulce",
     "forma" => "redondeada",
     "nombre"  => "manzana",
     3       => 4
);
```

Objetos

Para inicializar un objeto se utiliza el método *new* , y para acceder a cada uno de sus métodos se utiliza el operador -> .

```
class nada {
    function haz_nada () {
        echo "No estoy haciendo nada.";
    }
}

$miclase = new nada;
$miclase->haz_nada();
```

Conversión de Tipos de datos

Una variable en PHP, define su tipo según el contenido y el contexto en el que se utilice, es decir, si se asigna una hilera de texto a una variable, el tipo de esa variable será *string* . Si a esa misma variable se le asigna un número, el tipo cambiará a *entero* .

Para asegurarte de que una variable es del tipo adecuado se utiliza la función settype() . Para obtener el tipo de una variable se utiliza la función gettype() .

También es posible utilizar el mecanismo del *casting* tal y como se utiliza en C.

```
<html>
<head> <title>Ejemplo 6</title></head>
<body>
 <h1> Ejemplo de PHP </h1>
```

```
<?php

 $foo = 10;   // $foo es un entero
 $bar = (double) $foo;   // $bar es un doble

 #Mostramos resultados
 print "bar=$bar , foo=$foo <br>\n";

?>

</body>
</html>
```

Los tipos de casting permitidos son:

- (int), (integer) - fuerza a entero (integer)
- (real), (double), (float) - fuerza a doble (double)
- (string) - fuerza a hilera de texto (string)
- (array) - fuerza a array (array)
- (object) - fuerza a objeto (object)

1.1.2 Variables

En PHP las variables se representan como un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas. Las variables se asignan normalmente por valor, pero desde PHP4, también se asignan por referencia usando el símbolo &

```
<html>
<head> <title>Ejemplo 7</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php
 $foo = 'Bob';              // Asigna el valor 'Bob' a $foo
 $bar = &$foo;              // Referencia $foo vía $bar.
 $bar = "Mi nombre es $bar";  // Modifica $bar...
 echo $foo." <br>\n";                 // $foo también se modifica.
 echo $bar." <br>\n";
?>

</body>
</html>
```

Algo importante a tener en cuenta es que sólo las variables con nombre pueden ser asignadas por referencia.

Variables predefinidas

En PHP cada vez que se ejecuta un script, existen variables que se crean y que nos pueden informar del entorno en el que se está ejecutando dicho script.

Para obtener una lista de todas estas variables predefinidas se puede utilizar la funcion PHPinfo().

De todas estas variables, algunas se crean dependiendo del servidor que se esté utilizando y otras son propias de PHP.

Si se tratara de un servidor Apache, la lista de variables es:

- GATEWAY_INTERFACE:
- SERVER_NAME
- SERVER_SOFTWARE
- SERVER_PROTOCOL
- REQUEST_METHOD
- QUERY_STRING
- DOCUMENT_ROOT
- HTTP_ACCEPT
- HTTP_ACCEPT_CHARSET
- HTTP_ENCODING
- HTTP_ACCEPT_LANGUAJE
- HTTP_CONNECTION
- HTTP_HOST
- HTTP_REFERER
- HTTP_USER_AGENT
- REMOTE_ADDR
- REMOTE_PORT
- SCRIPT_FILENAME
- SERVER_ADMIN
- SERVER_PORT
- SERVER_SIGNATURE
- PATH_TANSLATED
- SCRIPT_NAME
- REQUEST_URL

las variables creadas por el propio PHP son:

- argv
- argc
- PHP_SELF
- HTTP_COOKIE_VARS
- HTTP_GET_VARS
- HTTP_POST_VARS

Nota: Esta lista no es exhaustiva ni pretende serlo. Simplemente es una guía de qué tipo de variables predefinidas se puede esperar tener disponibles en un script PHP.

Ámbito de una variable

El ámbito de una variable en PHP es exactamente igual que en C o en Perl tomando siempre en cuenta los archivos incluidos al principio de cada programa.

La única diferencia se encuentra en las variables globales, que tienen que ser expresamente definidas dentro de las funciones.

```
<html>
<head> <title>Ejemplo 8</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php
 $a = 1;
 $b = 2;

 Function Sum () {
     global $a, $b;

    $b = $a + $b;
}

Sum ();
echo $b;

?>

</body>
</html>
```

Variables variables

PHP permite un mecanismo para mantener variables con un nombre no fijo. Por ejemplo:

```
$a = "hola";
$$a = "mundo";
```

El ejemplo anterior, define dos variables, una denominada :math:‘a que contiene el valor «hola» y otra que se llama ‘hola que contiene el valor «mundo»

Para acceder al valor de una variable, se accede con:

```
echo "$a ${$a}";
```

La instrucción anterior provocará la salida «hola mundo».

Algo que se debe tener en cuenta cuando se utilizan variables, es que hay que resolver la ambiguedad que se crea al utilizar arreglos de variables de este tipo. Por ejemplo *:math:‘$a[1]* provoca una ambiguedad para el intérprete, puesto que no sabe si se desea utilizar la variable denominada *a[1] o utilizar la variables :math:‘a indexándola en su primer valor. Para esto se utiliza una sintaxis especial que sería* *{{$a}[1]* según se desee una opción u otra.

Variables de los formularios HTML

Cuando existe un formulario en HTML, inmediatamente después de ser enviado, dentro del ámbito PHP se crean automáticamente una variable por cada uno de los objetos que contiene el formulario.

Por ejemplo, consideremos el siguiente formulario:

```
<html>
<head> <title>Ejemplo 9</title></head>
<body>
 <h1> Ejemplo de Formulario 1 </h1>

<p>
Dame tu nombre !!!

<form action="ej10.php" method="post">
     Nombre: <input type="text" name="nombre">     <input type="submit">
</form>

</body>
</html>
```

Cuando es enviado, PHP creará la variable *$nombre*, que contendrá lo que sea que se introdujo en el campo Nombre:: del formulario.

```
<html>
<head> <title>Ejemplo 10</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php
 print "<h2>Hola $nombre </h2>\n";

?>

</body>
</html>
```

PHP también maneja arreglos en el contexto de variables de formularios, pero sólo en una dimensión. Se puede, por ejemplo, agrupar juntas variables relacionadas, o usar esta característica para recuperar valores de un campo select input múltiple:

```
<html>
<head> <title>Ejemplo 11</title></head>
<body>
 <h1> Ejemplo de Formulario 2 </h1>


<form action="ej12.php" method="post">
     Nombre: <input type="text" name="personal[name]">     E-mail: <input type="text"␣
↪name="personal[email]">     Cerveza: <br>
     <select multiple name="beer[]">
         <option value="warthog">Warthog
         <option value="guinness">Guinness
         <option value="stuttgarter">Stuttgarter Schwabenbr%u
     </select>
     <input type="submit">
 </form>
</body>
</html>


<html>
```

```
<head> <title>Ejemplo 12</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php

 print "<h2>Hola $personal[name] , ";
 print "tu email es $personal[email] y ";
 print "te gusta la cerveza $beer[0] </h2>\n";

?>

</body>
</html>
```

Si la posibilidad de PHP de track_vars está activada (se hace en la configurtación previa a la compilación), las variables enviadas con los métodos POST o GET también se encontrarán en los arreglos asociativos globales *:math:'HTTP_POST_VARS* y *'HTTP_GET_VARS*.

1.1.3 Constantes

Las constantes en PHP tienen que ser definidas por la función *'define() <php_manual_es.html#function.define>'__* y además no pueden ser redefinidas con otro valor.

Además, existen una serie de variables predefinidas denominadas:

- _FILE_: Fichero que se está procesando.
- _LINE_: Línea del fichero que se está procesando
- _PHP_VERSION: Versión de PHP.
- PHP_OS: Sistema operativo del cliente.
- TRUE: Verdadero.
- FALSE: Falso.
- E_ERROR: Error sin recuperación.
- E_WARNING: Error recuperable.
- E_PARSE: Error no recuperable (sintaxis).
- E_NOTICE: Puede Tratarse de un error o no. Normalmente permite continuar la ejecución.

Todas las constantes que empiezan por «E_»se utilizan normalmente con la función *error_reporting()*.

```
<html>
<head> <title>Ejemplo 14</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php
define("CONSTANTE", "hello world.");
echo CONSTANTE;

?>

</body>
</html>
```

1.1.4 Expresiones y operadores

En PHP una expresión es cualquier cosa que pueda contener un valor. Las expresiones más simples son las variables y las constantes y otras más complicadas serán las funciones, puesto que cada función devuelve un valor al ser invocada, es decir, contiene un valor, por lo tanto, es una expresión.

Todas las expresiones en PHP son exactamente igual que en C. Los operadores abreviados, los incrementos, etc, son exactamente iguales. Incluso existen otros operadores adicionales como el operador «.» que concatena valores de variables, o el operador «===» denominado operador de identidad que devolverá verdadero si las expresiones a ambos lados del operador contienen el mismo valor y a la vez son del mismo tipo. Por último, el operador «@» sirve para el control de errores. Para poder ver como funciona el operador @, veamos un ejemplo:

```
<?php
$res = @mysql\_query("select nombre from clientes")
   or die   ("Error en la selección, '$php\_errormsg'");
?>
```

Este ejemplo, utiliza el operador @ en la llamada a *mysql_query* y en el caso de dar un error, se salvará el mensaje devuelto en una variable denominada *php_errormsg*. Esta variable contendra el mensaje de error de cada instrucción y si ocurre otro error posterior, se machaca el valor con la nueva hilera de texto.

PHP mantiene también los operadores » “ » que sirven para ejecutar un comando del sistema tal y como hace la función *‘system() <php_manual_es.html#function.system>‘__*.

En PHP existen dos operadores *and* y dos operadores *or* que son: “and”, “&&” y “or”, “||” respectivamente, que se diferencian en la precedencia de cada uno.

La tabla que nos puede resumir la precedencia de cada uno de los operadores es:

Asocitividad	Operadores
Izquierda	,
Izquierda	or
Izquierda	xor
Izquierda	and
Derecha	print
Izquierda	= += -* *= /= .= %= &= |= ^= ~= <<= >>=
Izquierda	?:
Izquierda	||
Izquierda	&&
Izquierda	|
Izquierda	^
Izquierda	&
No posee	== != ===
No posee	< <= > >=
Izquierda	>> <<
Izquierda	▪ • .
Izquierda	* / %
Derecha	! ~ ++ – (int) (double) (string) (array) (object) @
Derecha	[
No posee	new

```
<html>
<head> <title>Ejemplo 15</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php

 function double($i) {
     return $i*2;
 }


 $b = $a = 5;        /* asignar el valor cinco a las variables $a y $b */
 $c = $a++;          /* postincremento, asignar el valor original de $a (5) a $c */
 $e = $d = ++$b;     /* preincremento, asignar el valor incrementado de $b (6) a
                        $d y a $e */

 /* en este punto, tanto $d como $e son iguales a 6 */
 $f = double($d++);  /* asignar el doble del valor de $d antes
                        del incremento, 2*6 = 12 a $f */
 $g = double(++$e);  /* asignar el doble del valor de $e después
                        del incremento, 2*7 = 14 a $g */
 $h = $g += 10;      /* primero, $g es incrementado en 10 y termina valiendo 24.
                        después el valor de la asignación (24) se asigna a $h,
                        y $h también acaba valiendo 24. */

 #Operador de ejecución
 $output = `ls -al`;
 echo "<pre>$output</pre><br>";


 echo "<h3>Postincremento</h3>";
 $a = 5;
 echo "Debería ser 5: " . $a++ . "<br>\n";
 echo "Debería ser 6: " . $a . "<br>\n";

 echo "<h3>Preincremento</h3>";
 $a = 5;
 echo "Debería ser 6: " . ++$a . "<br>\n";
 echo "Debería ser 6: " . $a . "<br>\n";

 echo "<h3>Postdecremento</h3>";
 $a = 5;
 echo "Debería ser 5: " . $a-- . "<br>\n";
 echo "Debería ser 4: " . $a . "<br>\n";

 echo "<h3>Predecremento</h3>";
 $a = 5;
 echo "Debería ser 4: " . --$a . "<br>\n";
 echo "Debería ser 4: " . $a . "<br>\n";
?>

</body>
</html>
```

1.1.5 Estructuras de Control

Además de la sintaxis normal (parecida al Perl o al C), PHP ofrece una sintaxis altenativa para alguna de sus estructuras de control; a saber, if, while, for, y switch. En cada caso, la forma básica de la sintaxis alternativa es cambiar abrir-llave por dos puntos (:) y cerrar-llave por endif;, endwhile;, endfor;, or endswitch;, respectivamente.

```
<html>
<head> <title>Ejemplo 16</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php


$a=8;
$b=6;

// Primer if
if ($a > $b) {
      print "a es mayor que b<br>";
      $b = $a;
  }

// if alternativo
if ($a > $b):
  print "A es mayor que B<br>";
endif;

// Segundo if (con else y elseif )
if ($a > $b) {
      print "a es mayor que b<br>";
  } elseif ($a == $b) {
      print "a es igual que b<br>";
  } else {
      print "b es mayor que a<br>";
  }

 // Segundo if alternativo
 if ($a > $b):
      print "A es mayor que B<br>";
      print "...";
 elseif ($a == $b):
      print "A es igual a B<br>";
      print "!!!";
 else:
      print "B es mayor que A<br>";
  endif;
?>

</body>
</html>
```

La mejor forma de resumir cada una de las opciones que ofrece PHP para las estructuras de control es mediante una tabla:

Estructura	Alternativa
If, if else, if elseif	if: endif;
while	while: endwhile;
for	for: endfor;
do.. while	▪
foreach(array as :math:‘value) - foreach(a rray as ‘key=>$value)	
switch	switch: endswitch;
continue	▪
break	▪
require()(Necesitan estar dentro de tags PHP)	▪
include()(Necesitan estar dentro de tags PHP)	▪

La instrucción require() se sustituye a sí misma con el archivo especificado, tal y como funciona la directiva #include de C. La instrucción include() incluye y evalúa el archivo especificado.

A diferencia de include(), require() siempre leerá el archivo referenciado, incluso si la línea en que está no se ejecuta nunca. Si se quiere incluir condicionalmente un archivo, se usa include(). La instrucción conditional no afecta a require(). No obstante, si la línea en la cual aparece el require() no se ejecuta, tampoco se ejecutará el código del archivo referenciado.

De forma similar, las estructuras de ciclo no afectan la conducta de require().. Aunque el código contenido en el archivo referenciado está todavía sujeto al ciclo, el propio require() sólo ocurre una vez. Esto significa que no se puede poner una instrucción require() dentro de una estructura de ciclo y esperar que incluya el contenido de un archivo distinto en cada iteración. Para hacer esto, usa una instrucción include(). Así, *require*, reemplaza su llamada por el contenido del fichero que requiere, e *include*, incluye y evalua el fichero especificado.

```
<?php
  print "Hola Mundo !<br>\n";
?>
```

El archivo que realiza la inclusión del primero sería algo similar a esto:

```
<html>
<head> <title>Ejemplo 18</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php include( 'ej17.php' ); ?>

</body>
</html>
```

1.1.6 Funciones

Funciones definidas por el usuario

Un ejemplo puede ser:

```
function foo($arg1, $arg2, ..., $argN) {
     echo "Función ejemplo";
     return $value;
}
```

Dentro de una función puede aparecer cualquier cosa, incluso otra función o definiciones de clase.

Respecto al paso de argumentos, son siempre pasados por valor y para pasarlos por referencia hay que indicarlo y se puede hacer de dos formas diferentes, en la definición de la función, anteponiendo el símbolo *&* al argumento que corresponda, en este caso la llamada será igual que la llamada a una función normal, o manteniendo la definición de la función normal y anteponer un *&* delante del argumento que corresponda en la llamada a la función.

```
<html>
<head> <title>Ejemplo 19</title></head>
<body>
 <h1> Ejemplo de PHP </h1>

<?php

//Define la función con parametros por referencia
function suma1 (&$a, &$b)
{
  $c=$a+$b;
  return $c;
}

//Define la función con parametros por valor
function suma2 ($a, $b)
{
  $c=$a+$b;
  return $c;
}

$a=2; $b=3; $suma;

//Llama la función 1 por referencia (no puede ser de otra forma)
print $suma=suma1($a,$b);

//Llama la función 2 por referencia
print $suma=suma1(&$a,&$b);

//Llama la función 2 por valor
print $suma=suma1($a,$b);

?>

</body>
</html>
```

PHP permite el mecanismo de argumentos por defecto. Un ejemplo de esta caracteristica es:

```
function hacerCafe($tipo="capuchino") {
     return "he hecho un café $tipo\n";
}
```

En la llamada a esta función se obtendrá una frase u otra según se llame:

```
echo hacerCafe();
echo hacerCafe("expreso");
```

En el caso de tratarse de una función con argumentos por defecto y argumentos normales, los argumentos por defecto deberán estar agrupados al final de la lista de argumentos.

En PHP4 el número de argumentos de una función definida por el usuario, puede ser variable, se utilizan las funciones func_num_args(), func_get_arg() y func_get_args().

Valores devueltos

A diferencia de C, PHP puede devolver cualquier número de valores, sólo hará falta recibir estos argumentos de la forma adecuada. Ejemplo:

```
function numeros() {
     return array(0,1,2);
}

list ($cero, $uno, $dos) = numeros();
```

Funciones variables

PHP soporta el concepto de funciones variable, esto significa que si una variable tiene unos paréntesis añadidos al final, PHP buscará una función con el mismo nombre que la evaluación de la variable, e intentará ejecutarla.

```
<?php
 function foo() {
     echo "En foo()<br\>\n";\
 }

 function bar ($arg ='') {
     echo " bar();El argumento ha sido '$arg'.<br\>\n";\
 }

 $func = 'foo';
 $func();
 $func='bar';
 $func('test');
?>
```

1.2 Procesamiento de formularios

Es muy sencillo procesar formularios con PHP, ya que los parámetros del formulario están disponibles en los arreglos _GET y _POST.

1.2.1 Métodos

Existen dos métodos HTTP que un cliente puede utilizar para pasar los datos del formulario al servidor: GET y POST. El método que utiliza un formulario particular, se especifica con el atributo *method* en la etiqueta *form*. En teoría, los métodos son sensibles a mayúsculas en el código HTML, pero en la práctica algunos navegadores fallan si el el nombre del método no está en mayúsculas.

A solicitud GET codifica los parámetros del formulario en la dirección URL en lo que se llama una cadena de consulta, el texto que sigue al carácter *?* es la cadena de consulta:

```
/path/to/page.php?keyword=bell&length=3
```

Una solicitud POST pasa los parámetros del formulario en el cuerpo de la solicitud HTTP, dejando intacta la URL. El tipo de método que se utilizó para solicitar una página PHP está disponible a través de $_SERVER[“REQUEST_METHOD”]. Por ejemplo:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
  // handle a GET request
} else {
  die("You may only GET this page.");
}
```

1.2.2 Parámetros

Se utilizan los arreglos _POST, _GET y _FILES para acceder a los parámetros de formulario desde el código PHP. Las llaves son los nombres de los parámetros y los valores son los valores de esos parámetros. Por ejemplo, considere la siguiente página utilizada para separar una palabra:

```
<html>
<head><title>Formulario</title></head>
<body>
    <form action="separar.php" method="POST">
        Ingrese una palabra:
        <input type="text" name="word"/><br/>
        Largo de las separaciones:
        <input type="text" name="number" /><br/>
        <input type="submit" value="Dividir">
    </form>
</body>
</html>
```

El programa PHP para procesar dicho formulario sería el siguiente:

```
$word = $_POST['word'];
$number = $_POST['number'];
$chunks = ceil(strlen($word) / $number);
echo "The {$number}-letter chunks of '{$word}' are:<br />\n";
for ($i = 0; $i < $chunks; $i++) {
  $chunk = substr($word, $i * $number, $number);
  printf("%d: %s<br />\n", $i + 1, $chunk);
}
```

1.2.3 Páginas con auto-procesamiento

Una página PHP puede ser utilizada tanto para generar un formulario como para procesarlo.

```
<html>
<head><title>Temperature Conversion</title></head>
<body>
    <?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
        <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST">
            Fahrenheit temperature:
            <input type="text" name="fahrenheit" /><br/>
            <input type="submit" value="Convert to Celsius!" />
        </form>
        <?php } else if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $fahrenheit = $_POST['fahrenheit'];
            $celsius = ($fahrenheit - 32) * 5 / 9;
            printf("%.2fF is %.2fC", $fahrenheit, $celsius);
        } else {
            die("This script only works with GET and POST requests.");
        } ?>
    </body>
</html>
```

Otra forma de programa decide si se debe mostrar un formulario o proceso es ver si alguno de los parámetros se ha suministrado. Esto le permite escribir una página de auto-procesamiento que utiliza el método GET para enviar valores.

```
<html>
  <head><title>Temperature Conversion</title></head>
  <body>
    <?php $fahrenheit = $_GET['fahrenheit'];
      if (is_null($fahrenheit)) { ?>
    <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
      Fahrenheit temperature:
      <input type="text" name="fahrenheit" /><br />
      <input type="submit" value="Convert to Celsius!" />
    </form>
   <?php } else {
     $celsius = ($fahrenheit - 32) * 5 / 9;
     printf("%.2fF is %.2fC", $fahrenheit, $celsius); } ?>
  </body>
</html>
```

1.2.4 Formularios adhesivos

Muchos sitios web utilizan una técnica conocida como formularios adhesivos, en el que los resultados de una consulta se acompañan de un formulario de búsqueda cuyos valores por defecto son los de la consulta anterior.

La técnica básica consiste en utilizar el valor enviado por el formulario como el valor por defecto cuando se crea el campo HTML.

```
<html>
  <head><title>Temperature Conversion</title></head>
  <body>
    <?php $fahrenheit = $_GET['fahrenheit']; ?>
   <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
     Fahrenheit temperature:
     <input type="text" name="fahrenheit" value="<?php echo $fahrenheit; ?>" /><br/>
     <input type="submit" value="Convert to Celsius!" />
   </form>
```

```
   <?php if (!is_null($fahrenheit)) {
     $celsius = ($fahrenheit - 32) * 5 / 9;
     printf("%.2fF is %.2fC", $fahrenheit, $celsius);
   } ?>
  </body>
</html>
```

1.2.5 Parámetros multivaluados

Las listas de selección HTML, creadas con la etiqueta *select*, pueden permitir selecciones múltiples. Para asegurarse de que PHP reconoce los múltiples valores que el navegador pasa a un programa de procesamiento de formularios, es necesario hacer que el nombre del campo en la formulario HTML finalice *[]*.

```
<html>
  <head><title>Personality</title></head>
  <body>
    <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
      Select your personality attributes: <br/>
      <select name="attributes[]" multiple>
        <option value="perky">Perky</option>
        <option value="morose">Morose</option>
        <option value="thinking">Thinking</option>
        <option value="feeling">Feeling</option>
        <option value="thrifty">Spend-thrift</option>
        <option value="shopper">Shopper</option>
      </select><br/>
      <input type="submit" name="s" value="Record my personality!" />
    </form>
<?php if (array_key_exists('s', $_GET)) {
   $description = join(' ', $_GET['attributes']);
   echo "You have a {$description} personality.";
} ?>
  </body>
</html>
```

Otro ejemplo similar pero que utiliza *checkboxes* es:

```
<html>
  <head><title>Personality</title></head>
  <body>
    <form action="<?php $_SERVER['PHP_SELF']; ?>" method="GET">
      Select your personality attributes:<br />
      <input type="checkbox" name="attributes[]" value="perky" /> Perky<br />
      <input type="checkbox" name="attributes[]" value="morose" /> Morose<br />
      <input type="checkbox" name="attributes[]" value="thinking" /> Thinking<br />
      <input type="checkbox" name="attributes[]" value="feeling" /> Feeling<br />
      <input type="checkbox" name="attributes[]" value="thrifty" />Spend-thrift<br />
      <input type="checkbox" name="attributes[]" value="shopper" /> Shopper<br /><br /
↪>
      <input type="submit" name="s" value="Record my personality!" />
    </form>
<?php if (array_key_exists('s', $_GET)) {
  $description = join (' ', $_GET['attributes']);
  echo "You have a {$description} personality.";
} ?>
  </body>
```

```
</html>
```

1.2.6 Parámetros multivaluados adhesivos

Para manejar parámetros multivaluados adhesivos es útil escribir una función para generar el código HTML de los valores posibles y trabajar a partir de una copia de los parámetros enviados.

```
<html>
  <head><title>Personality</title></head>
  <body>

<?php
  $attrs = $_GET['attributes'];
  if (!is_array($attrs)) {
    $attrs = array();
}

function makeCheckboxes($name, $query, $options) {
  foreach ($options as $value => $label) {
    $checked = in_array($value, $query) ? "checked" : '';
    echo "<input type=\"checkbox\" name=\"{$name}\"
          value=\"{$value}\" {$checked} />";
    echo "{$label}<br />\n"; }
  }

$personalityAttributes = array(
  'perky'=> "Perky",
  'morose'=> "Morose",
  'thinking'=> "Thinking",
  'feeling'=> "Feeling",
  'thrifty'=> "Spend-thrift",
  'prodigal'=> "Shopper"
); ?>

  <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
    Select your personality attributes:<br />
    <?php makeCheckboxes('attributes', $attrs, $personalityAttributes); ?><br />
    <input type="submit" name="s" value="Record my personality!" />
  </form>

<?php if (array_key_exists('s', $_GET)) {
  $description = join (' ', $_GET['attributes']);
  echo "You have a {$description} personality.";
} ?>

  </body>
</html>
```

1.3 Manteniendo el estado

HTTP es un protocolo sin estado, lo que significa que una vez que un servidor web completa la petición de un cliente para una página web, la conexión entre los dos desaparece. En otras palabras, no hay manera en que un servidor pueda reconocer que toda una secuencia de peticiones se originan desde el mismo cliente.

Para solucionar esta falta de estado del protocolo HTTP, existen varias técnicas para realizar un seguimiento de la información de estado entre las solicitudes (también conocido como el seguimiento de sesión).

1.3.1 Campos ocultos en formularios

Una de estas técnicas es el uso de campos de formulario ocultos para pasar la información. PHP trata a los campos de formulario ocultos al igual que los campos de formulario normales, por lo que los valores están disponibles en los arreglos :math:‘_GET y ‘_POST. Mediante el uso de campos de formulario ocultos, se puede mantener toda la información que fuera necesaria para cualquier aplicación. Sin embargo, una técnica más común consiste en asignar a cada usuario un identificador único y pasar el identificador usando un único campo de formulario oculto. Mientras que los campos de formulario ocultos funcionan en todos los navegadores, ellos trabajan sólo para una secuencia de formularios generados de forma dinámica, por lo que generalmente no son tan útiles como algunas otras técnicas.

```
<?php
  $number = $_POST['number'];
  if (isset($number)) {
     $count = intval($_POST['count']);
     $count++;
     $numbers = Array();
     array_push($numbers,$number);
     for ($i = 0; $i < $count-1; $i++) {
        array_push($numbers,$_POST['number'.$i]);
    }
  } else {
    $count = 0;
  }
?>
<html>
    <head>
        <title>My Lottery</title>
    </head>
    <body>
   <h2>My Lottery</h2>
   <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
   <?php
      if ($count == 0) {
        echo "<h3>Wellcome!!</h3>";
      } else {
        echo "<label>Your winning numbers are: </label>";
        for ($i = 0; $i < $count-1; $i++)
            echo "<b>".$numbers[$i]."</b>, ";
        echo "<b>".$numbers[$count-1]."</b></p>";
      }
      if ($count == 6) {
          echo "<h3>Good luck!!</h3>";
      } else {
    ?>
   <label>Please, enter a number:</label>
   <input type='text' name='number'/>
   <input type='submit'>
   <?php } ?>
   <input type="hidden" value="<?php echo $count; ?>" name="count"/>
   <?php
      for ($i = 0; $i < $count; $i++) { ?>
      <input type="hidden" value="<?php echo $numbers[$i]; ?>"
             name="number<?php echo $i?>"/>
   <?php } ?>
```

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

1.3.2 Cookies

La segunda y más amplia técnica de mantener el estado es el uso de cookies. Una cookie es un fragmento de información que el servidor puede dar a un cliente. En cada solicitud posterior el cliente le dará esa información de vuelta al servidor, identificando así a sí mismo. Las cookies son útiles para conservar la información a través de las repetidas visitas de un navegador, pero también tienen sus propios problemas. El principal problema es que la mayoría de los navegadores permiten a los usuarios desactivar las cookies. Por lo que cualquier aplicación que utiliza las cookies para el mantenimiento del estado tiene que usar otra técnica como un mecanismo de reserva.

Una cookie es básicamente una cadena que contiene varios campos. Un servidor puede enviar una o más cookies a un navegador en las cabeceras de una respuesta. Algunos de los campos de la galleta indican las páginas para las que el navegador debe enviar la cookie como parte de la solicitud.

PHP soporta cookies HTTP de forma transparente. Se pueden configurar cookies usando la función *setcookie()* o *setrawcookie()*. Las cookies son parte del header HTTP, así es que *setcookie()* debe ser llamada antes que cualquier otra salida sea enviada al browser. Los envíos de cookies desde el cliente serán incluidos automáticamente en el arreglo $_COOKIE.

```
<?php
  $number = $_POST['number'];
  if (isset($number)) {
     $count = intval($_COOKIE['count']);
     setcookie('number'.$count,$number);
     $count++;
  } else {
    foreach ($_COOKIE as $key => $value )
      setcookie($key, FALSE);
    $count = 0;
  }
  setcookie('count', $count);
?>
<html>
    <head>
        <title>My Lottery</title>
    </head>
    <body>
       <h2>My Lottery</h2>
       <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
       <?php
          if ($count == 0) {
            echo "<h3>Wellcome!!</h3>";
          } else {
            echo "<label>Your winning numbers are: </label>";
            for ($i = 0; $i < $count-1; $i++)
                echo "<b>".$_COOKIE['number'.$i]."</b>, ";
            echo "<b>$number</b></p>";
          }
          if ($count == 6) {
              echo "<h3>Good luck!!</h3>";
          } else {
        ?>
       <label>Please, enter a number:</label>
```

```
       <input type='text' name='number'/>
       <input type='submit'>
       <?php } ?>
       </form>
    </body>
</html>
```

1.3.3 Uso de sesiones

La mejor manera de mantener el estado con PHP es utilizar el sistema integrado de seguimiento de sesiones. Este sistema permite crear variables persistentes que son accesibles desde diferentes páginas de la aplicación, así como en diferentes visitas al sitio por el mismo usuario. Internamente, el mecanismo de seguimiento de la sesión de PHP utiliza cookies (o URLs) para resolver con elegancia la mayoría de los problemas que requieren del estado, cuidando de todos los detalles para el programador.

La función PHP *session_start* crea una sesión o reanuda la actual basada en un identificador de sesión pasado mediante una petición GET o POST, o pasado mediante una cookie. El soporte para sesiones permite almacenar los datos entre peticiones en el arreglo $_SESSION. La función *session_destroy* destruye toda la información asociada con la sesión actual. No destruye ninguna de las variables globales asociadas con la sesión, ni destruye la cookie de sesión.

```
<?php
  session_start();
  $number = $_POST['number'];
  if (isset($number)) {
     $count = intval($_SESSION['count']);
     $_SESSION['number'.$count] = $number;
     $count++;
  } else {
    session_destroy();
    $count = 0;
  }
  $_SESSION['count'] = $count;
?>
<html>
    <head>
        <title>My Lottery</title>
    </head>
    <body>
       <h2>My Lottery</h2>
       <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
       <?php
          if ($count == 0) {
            echo "<h3>Wellcome!!</h3>";
          } else {
            echo "<label>Your winning numbers are: </label>";
            for ($i = 0; $i < $count-1; $i++)
                echo "<b>".$_SESSION['number'.$i]."</b>, ";
            echo "<b>$number</b></p>";
          }
          if ($count == 6) {
              echo "<h3>Good luck!!</h3>";
          } else {
        ?>
       <label>Please, enter a number:</label>
       <input type='text' name='number'/>
       <input type='submit'>
```

```
        <?php } ?>
        </form>
    </body>
</html>
```

1.3.4 Reescritura del URL

Otra técnica es la reescritura de URL, donde cada URL local en el que el usuario puede hacer clic se modifica dinámicamente para incluir información adicional. Esta información adicional se suele especificar como un parámetro en la URL. Por ejemplo, si se asigna a cada usuario un identificador único, es posible incluir ese ID en todas las direcciones URL, de la siguiente manera:

```
http://www.example.com/catalog.php?userid=123
```

Si es posible modificar dinámicamente todos los enlaces locales para incluir un ID de usuario, se podrá realizar un seguimiento de los usuarios individuales en su aplicación. La reescritura de URL trabaja para todos los documentos generados dinámicamente, y no sólo los formularios, pero en realidad llevar a cabo la reescritura puede ser tedioso.

1.3.5 Ejemplo

A continuación se muestra un ejemplo programado de una aplicación Web en PHP que permite llevar una lista de eventos. Cada evento tiene una fecha, hora, y asunto. Se pueden agregar nuevos eventos o eliminar los existentes.

En este caso se utilizan «cookies» para resolver el problema. Note como se crea un hilera de texto (string) con los diferentes campos a almacenar utilizando como delimitador el carácter "|", y luego se genera otra hilera de texto con

todos los eventos del arreglo pero ahora delimitados por los caracteres "/n". Esta última hilera de texto es la que se almacena en la "cookie".

```
<?php
  if (isset($_COOKIE['eventos'])) {
    $array = explode("/n",$_COOKIE['eventos']);
  }  else {
    $array = array();
  };

  if (isset($_POST['borrar'])) {
      $id = $_POST['borrar'];
      unset($array[$id]);
      $array = array_values($array);
      setcookie('eventos',implode("/n",$array));
  } else if (isset($_POST['agregar'])) {
      $new_item = $_POST['dia'].'|'.$_POST['hora'].'|'.$_POST['evento'];
      $array[] = $new_item;
      setcookie('eventos',implode("/n",$array));
  } else {
      setcookie('eventos',null);
      $array = array();
  }
?>
<html>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <head>
        <title>Calendario de eventos</title>
    </head>
    <body>
      <h2>Calendario de eventos</h2>
      <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
        <table border=1>
            <tr><th>Día</th><th>Hora</th><th>Evento</th><th>Operación</th></tr>
            <?php
              for ($i=0;$i<sizeof($array);$i++) {
                $values = explode("|",$array[$i]);
                echo '<tr><td>'.$values[0].'</td><td>'.$values[1].'</td><td>'.
                    $values[2].'</td><td><button name="borrar" value="'.$i.
                    '">Borrar</button></td></tr>';
              }
            ?>
            <tr><td><input size="10" name="dia" type="date"/></td>
                <td><input name="hora" size="10" type="time"/></td>
                <td><input size="40" name="evento"/></td>
                <td><button name="agregar">Agregar</button></td>
            </tr>
        </table>
      </form>
    </body>
</html>
```

1.3.6 Ejercicios

1. Intente modificar el ejemplo anterior para manejar los eventos mediante campos ocultos.
2. Ahora, intente modificar el mismo ejemplo para almacenar los objetos mediante variables de sesión.

1.4 Manejo de archivos

La función *fopen(path,mode)* permite abrir un archivo local o mediante un URL. El *path* del archivo debe incluir la ruta completa al mismo. El *mode* puede ser *r* - lectura,*w* - escritura,*a* - agregar, o *x* - escritura exclusiva. Se puede agregar un + al modo y si el archivo no existe, se intentará crear. La función *fclose(file)* cierra un puntero a un archivo abierto.

La función *feof(file)* comprueba si el puntero a un archivo se encuentra al final del archivo. La función *fgets(file)* obtiene una línea desde el puntero a un archivo. La función *file_exists(file)* comprueba si existe un archivo o directorio.

```
<?php

$path = "/home/user/file.txt";
if (!file_exists($path))
    exit("File not found");
$file = fopen($path, "r");
if ($file) {
    while (($line = fgets($file)) !== false) {
        echo $line;
    }
    if (!feof($file)) {
        echo "Error: EOF not found\n";
    }
    fclose($file);
}

?>
```

La función *fscanf* analiza la entrada desde un archivo de acuerdo a un formato. Los tipos más importantes son: *%d* - entero, *%f* - flotante, y *%s* - string. Un detalle importante es que *%s* no reconoce hileras de texto con espacios en blanco, únicamente palabras completas.

```
<?php

$path = "/home/usr/data.txt";
if (!file_exists($path))
    exit("File not found");
$file = fopen($path, "r");
echo "<html><body><table border=1>";
echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th></tr>";
while ($data = fscanf($file, "%s\t%d\t%d\t%f\n")) {
    list ($country, $area, $pop, $dens) = $data;
    echo "<tr><td>".$country."</td><td>".$area."</td><td>".
         $pop."</td><td>".$dens."</td></tr>";
}
echo "</table></body></html>";
fclose($file);

?>
```

El archivo de datos para el ejemplo anterior podría ser el siguiente. Note que debe haber un tabulador que separe cada campo de un mismo registro.

```
Belice  22966   334000  14.54
Costa_Rica  51100   4726000 92.49
El_Salvador 21041   6108000 290.29
Guatemala   108894  15284000    140.36
```

```
Honduras    112492  8447000 75.09
Nicaragua   129494  6028000 46.55
Panama  78200   3652000 46.70
```

1.4.1 Directorios

La función *is_dir* indica si el nombre del archivo es un directorio y *is_file* indica si el nombre de archivo es realmente un archivo. La función *mkdir* crea un directorio. La función *rename* renombre un archivo o directorio. La función *rmdir* remueve un directorio y la función *unlink* remueve un archivo.

1.4.2 Archivos binarios

Las funciones *fread* y *fwrite* leen y escriben, respectivamente, en un archivo en modo binario. La función *fseek* posiciona el puntero del archivo.

```
<?php

$path = "/home/usr/data2.txt";
if (!file_exists($path))
    exit("File not found");
$file = fopen($path, "r");
echo "<html><body><table border=1>";
echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th></tr>";
fseek($file,35);
while ($data = fread($file, 35)) {
    $fields = explode("|",$data);
    echo "<tr><td>".$fields[0]."</td><td>".$fields[1]."</td><td>".
         $fields[2]."</td><td>".$fields[3]."</td></tr>";
}
echo "</table></body></html>";
fclose($file);

?>
```

El archivo de datos para el ejemplo anterior podría ser el siguiente. Note que este es un archivo de registros de tamaño fijo. Además, tome en cuenta que es necesario omitir los encabezados del archivo.

```
CountryName|Area  |People  |Densi.
Belice     | 22966|  334000| 14.54
Costa Rica | 51100| 4726000| 92.49
El Salvador| 21041| 6108000|290.29
Guatemala  |108894|15284000|140.36
Honduras   |112492| 8447000| 75.09
Nicaragua  |129494| 6028000| 46.55
Panama     | 78200| 3652000| 46.70
```

1.4.3 Archivos de texto

Otra forma de leer archivos de texto es utilizar la función *file*, la cual transfiere un archivo completo a un arreglo. Note que no es necesario abrir el archivo (*fopen*) para utilizar este función.

```
<?php

$path = "/home/usr/data2.txt";
if (!file_exists($path))
    exit("File not found");
$rows = file($path);
array_shift($rows);
echo "<html><body><table border=1>";
echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th></tr>";
foreach ($rows as $row) {
    $fields = explode("|",$row);
    echo "<tr><td>".$fields[0]."</td><td>".$fields[1]."</td><td>".
         $fields[2]."</td><td>".$fields[3]."</td></tr>";
}
echo "</table></body></html>";

?>
```

1.4.4 Archivos CSV

La función *fgetcsv* obtiene una línea del puntero a un archivo y la examina para tratar campos CSV. La función *fputcsv* da formato a una línea como CSV y la escribe en un puntero a un archivo.

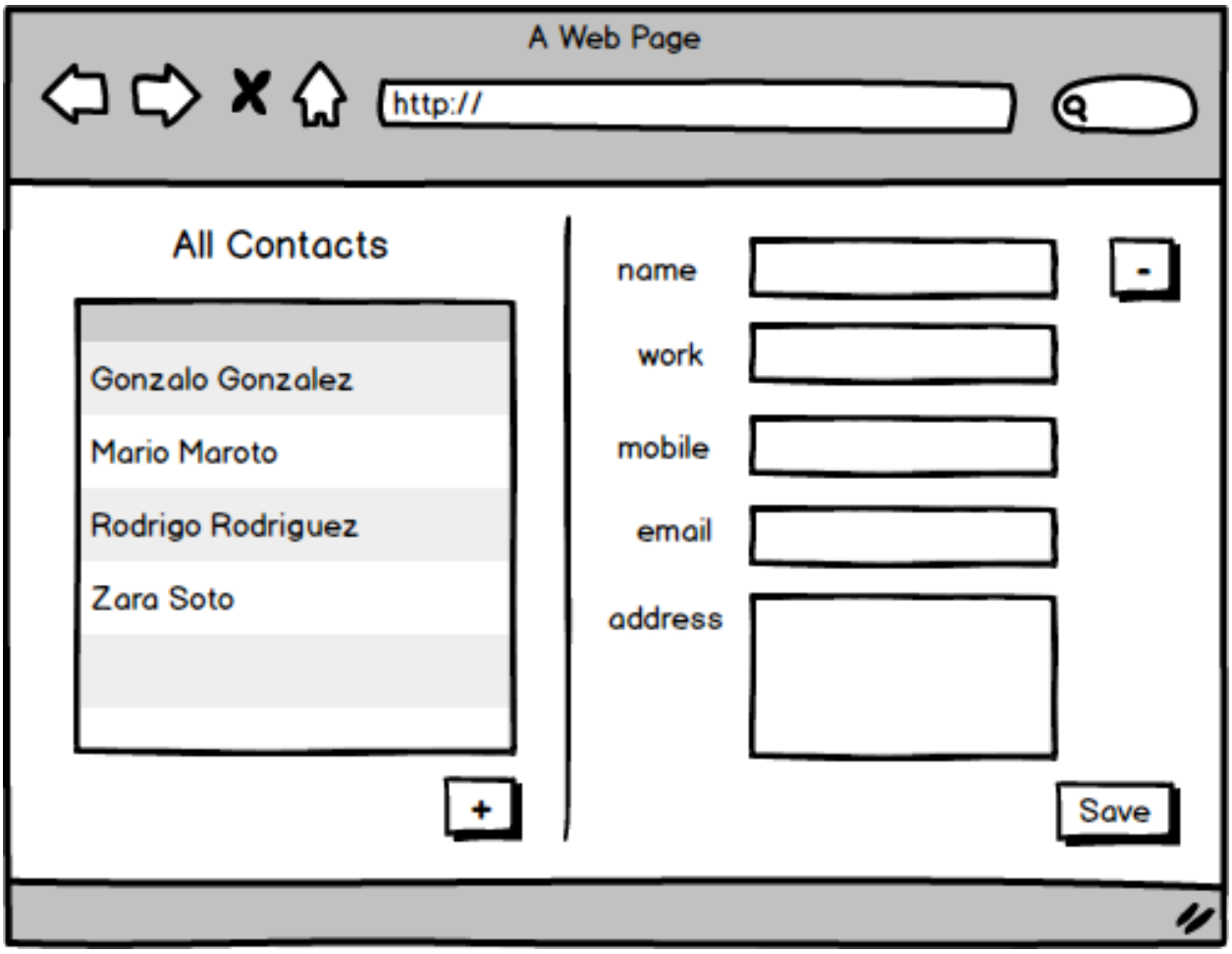
```
<?php

$path = "/home/usr/data3.csv";
if (!file_exists($path))
    exit("File not found");
$file = fopen($path, "r");
echo "<html><body><table border=1>";
echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th></tr>";
while ($fields = fgetcsv($file,",")) {
    echo "<tr><td>".$fields[0]."</td><td>".$fields[1]."</td><td>".
         $fields[2]."</td><td>".$fields[3]."</td></tr>";
}
echo "</table></body></html>";
fclose($file);

?>
```

El archivo de datos para el ejemplo anterior podría ser el siguiente.

```
Belice,22966,334000,14.54
Costa Rica,51100,4726000,92.49
El Salvador,21041,6108000,290.29
Guatemala,108894,15284000,140.36
Honduras,112492,8447000,75.09
Nicaragua,129494,6028000,46.55
Panama,78200,3652000,46.70
```

1.4.5 Ejemplo

A continuación se muestra un ejemplo de una aplicación Web en PHP que permite llevar una lista de contactos. Cada contacto tiene un nombre, teléfono del trabajo, teléfono móvil, dirección electrónica y dirección postal. Se pueden agregar nuevos contactos, modificar los existentes, o eliminarlos.


A Web Page
http://
All Contacts
Gonzalo Gonzalez
Mario Maroto
Rodrigo Rodriguez
Zara Soto
+
name
work
mobile
email
address
-
Save

Note que se utiliza *borrado perezoso* en esta solución, es decir, se marcan los registros borrados y no se eliminan realmente del archivo. También, en este ejemplo se utiliza un archivo con registros de tamaño fijo delimitados por el carácter "|".

```
<?php
$path = "data.txt";
if (file_exists($path))
  $file = fopen($path, "r+");
else
  $file = fopen($path, "a+");
while ($data = fread($file,154)) {
  $array[] = explode('|',$data);
};

if (isset($_GET['get'])) {
  $curr = (int)$_GET['get'];
  $item = $array[$curr];
} else if (isset($_GET['delete'])) {
  $curr = (int)$_GET['delete'];
  fseek($file,$curr*154,SEEK_SET);
  fwrite($file,'**deleted**');
  $array[$curr][0] = '**deleted**';
  $item = array('','','','','');
  $curr = 0;
} else if (isset($_GET['save'])) {
    $curr = (int)$_GET['save'];
    $item = array(str_pad($_GET['name'],30),
                  str_pad($_GET['work'],30),
                  str_pad($_GET['mobile'],30),
                  str_pad($_GET['email'],30),
                  str_pad($_GET['address'],30));
    fseek($file,$curr*154,SEEK_SET);
    fwrite($file,implode('|',$item));
    $array[$curr] = $item;
} else if (isset($_GET['append'])) {
  $item = array('','','','','');
  $curr = sizeof($array);
};

fclose($file);
?>
<html>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <head>
        <title>Contacts</title>
    </head>
    <body>
     <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
      <div style="width: 250px; float: left;">
        <h3>All Contacts</h3>
        <table width="150" border=1>
          <?php
            for ($i=0;$i<sizeof($array);$i++) {
              if (trim($array[$i][0])!="**deleted**")
                echo '<tr><td><a href="?get='.$i.
                     '" style="text-decoration:none;">'.
                     $array[$i][0].'</a></td></tr>';
            }
```

```
          ?>
        </table>
        </p>
        <button name="append" value="">+</button>
      </div>
      <div style="margin-left:250px;">
          <table>
            <tr><td> </td></tr>
            <tr><td><label>name</label></td><td>
                <input name="name" size="30" value="<?php echo $item[0]; ?>"/></td></
↪tr>
            <tr><td><label>work</label></td><td>
                <input name="work" size="30" value="<?php echo $item[1]; ?>"/></td></
↪tr>
            <tr><td><label>mobile</label></td><td>
                <input name="mobile" size="30" value="<?php echo $item[2]; ?>"/></td>
↪</tr>
            <tr><td><label>email</label></td><td>
                <input name="email" size="30" value="<?php echo $item[3]; ?>"/></td></
↪tr>
            <tr><td><label>address</label></td><td>
                <input name="address" size="30" value="<?php echo $item[4]; ?>"/></td>
↪</tr>
            <tr><td> </td></tr>
            <tr><td><button name="delete" value="<?php echo $curr; ?>">-</button></td>
                <td align="right">
                <button name="save" value="<?php echo $curr; ?>">Save</button></td></
↪tr>
          </table>
        </div>
      </form>
    </body>
</html>
```

1.4.6 Ejercicios

1. Intente modificar el ejemplo anterior para que utilice dos archivos, uno que almacene el nombre del contacto y un índice. Este índice indicará la posición en otro archivo en donde se encontrará el detalle del contacto.
2. Ahora, intente modificar el ejemplo para resolver el mismo problema pero utilizando registros de tamaño variable. Trate de solucionar de una forma "elegante" el borrado de registros.
3. Debido a que cada vez que se ejecuta la aplicación es necesario cargar todo el archivo en memoria, una mejor solución sería "paginar" los registros, es decir, cargar solo una pequeña parte en memoria y permitir que el usuario cargara más registros conforme los necesite (posiblemente mediante un par de botones adicionales).

1.5 Bases de datos con PDO

La extensión PDO (PHP Data Objects) de PHP consiste de una capa de abstracción para acceder a diferentes tipos de bases de datos. Utilizando PDO se logran estandarizar los diferentes mecanismos para realizar la conexión a una base de datos, así como recuperar y modificar información. Sin embargo, PDO no estandariza SQL lo que significa que se debe lidiar con las diferentes sintaxis de las instrucciones en cada administrador de bases de datos.

1.5.1 Manejadores de bases de datos

Para cada base de datos existe un manejador (driver) específico, que debe estar habilitado en el archivo de configuración de PHP (el archivo *php.ini*). Los manejadores se administran mediante extensiones de PHP, las cuales tienen nombres finalizando con *dll* en Windows y con *so* en Unix.

```
extension=php_pdo.dll
extension=php_pdo_firebird.dll
extension=php_pdo_informix.dll
extension=php_pdo_mssql.dll
extension=php_pdo_mysql.dll
extension=php_pdo_oci.dll
extension=php_pdo_oci8.dll
extension=php_pdo_odbc.dll
extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
```

Todas estas extensiones deben existir en el directorio de *extensiones* de PHP. Generalmente las extensiones *php_pdo* y *php_pdo_sqlite* estarán habilitadas por omisión.

1.5.2 Conexiones

Para realizar una nueva conexión se debe crear una instancia del objeto *PDO*. Este constructor acepta una serie de parámetros de conexión (string de conexión) que pueden ser específicos para cada sistema de bases de datos.

Si no se logra establecer la conexión se producirá una excepción (PDOException). Si la conexión es exitosa, una instancia de *PDO* será devuelta. La conexión permanece activa por todo le ciclo de vida del objeto *PDO*. Para cerrar la conexión, se debe destruir el objeto asegurándose que toda referencia sea eliminada, o bien, PHP cerrará la conexión automáticamente cuando el programa finalice.

Si se desea hacer una conexión persistente, que no sea eliminada al final de la ejecución del programa, es necesario habilitar la opción *PDO:ATTR_PERSISTENT* en el arreglo de las opciones de la conexión.

```
<?php
try {
    $dbh = new PDO('sqlite:test.db');
    $dbh->exec("CREATE TABLE countries
                (name TEXT, area INTEGER, population INTEGER, density REAL)");
    $dbh = null;
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Note que en el ejemplo anterior la base de datos podría ser creada usando el comando *sqlite3 test.db «»* (si está disponible en el ambiente). Además, el archivo *test.db* como el directorio en que se encuentra, deben tener derechos de escritura.

1.5.3 Transacciones

Debido a que no todas las bases de datos soportan transacciones, PHP corre en el modo de *auto-commit* que ejecuta cada instrucción individual en forma implícita. Si se desea usar transacciones, y no se desea utilizar el modo de *auto-commit*, es necesario invocar el método *PDO::beginTransaction()* al inicio de la transacción. Si el manejador de la base de datos no permite el uso de transacciones se producirá una excepción (*PDOException*). Cuando se acabe

de especificar la transacción se pueden utilizar los métodos *PDO::Commit* para aplicar dichas instrucciones, o bien, *PDO::rollBack* para abortar dicha transacción.

```
<?php
try {
  $dbh = new PDO('sqlite:test.db');
  echo "Connected\n";
} catch (Exception $e) {
  die("Unable to connect: " . $e->getMessage());
}

try {

  $dbh->beginTransaction();
  $dbh->exec("INSERT INTO countries (name, area, population, density)
                          values ('Belice',22966,334000,14.54)");
  $dbh->exec("INSERT INTO countries (name, area, population, density)
                          values ('Costa Rica',51100,4726000,92.49)");
  $dbh->exec("INSERT INTO countries (name, area, population, density)
                          values ('El Salvador',21041,6108000,290.29)");
  $dbh->exec("INSERT INTO countries (name, area, population, density)
                          values ('Guatemala',108894,15284000,140.36)");
  $dbh->exec("INSERT INTO countries (name, area, population, density)
                          values ('Honduras',112492,8447000,75.09)");
  $dbh->commit();

} catch (Exception $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
?>
```

Si una transacción no fue terminada con la instrucción *commit* y el programa finaliza, la base de datos abortará la transacción automáticamente.

1.5.4 Instrucciones preparadas

Una *instrucción preparada* es un tipo de plantilla para SQL que puede ser personalizada utilizando parámetros. Existen dos beneficios de utilizar *instrucciones preparadas* : la base de datos únicamente compilará una vez la instrucción lo cual ahorra mucho tiempo, y los parámetros no necesitan *comillas* ya que el manejador se encarga de agregarlas a la instrucción. El realizar el enlace (bind) de parámetros se puede realizar por mediante el nombre del parámetro o por posición (utilizando el símbolo ?).

```
<?php
try {
  $dbh = new PDO('sqlite:test.db');
  echo "Connected\n";
} catch (Exception $e) {
  die("Unable to connect: " . $e->getMessage());
}

try {
  $stmt = $dbh->prepare("INSERT INTO countries (name, area, population, density)
                                 VALUES (:name, :area, :population, :density)");
  $stmt->bindParam(':name', $name);
  $stmt->bindParam(':area', $area);
  $stmt->bindParam(':population', $population);
```

```
  $stmt->bindParam(':density', $density);

  $dbh->beginTransaction();
  $name = 'Nicaragua'; $area = 129494; $population = 602800; $density = 46.55;
  $stmt->execute();
  $name = 'Panama'; $area = 78200; $population = 3652000; $density = 46.70;
  $stmt->execute();
  $dbh->commit();

} catch (Exception $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
?>
```

Adicionalmente, es posible utilizar un arreglo para pasar los parámetros de la consulta. En este caso no es necesario incluir el enlace (bind) de parámetros. Es importante notar que el orden de los parámetros resulta vital aquí.

```
<?php
try {
  $dbh = new PDO('sqlite:test.db');
  echo "Connected\n";
} catch (Exception $e) {
  die("Unable to connect: " . $e->getMessage());
}

try {
  $stmt = $dbh->prepare("INSERT INTO countries (name, area, population, density)
                                VALUES (?, ?, ?, ?)");

  $dbh->beginTransaction();
  $stmt->execute(array('Nicaragua', 129494, 602800, 46.55));
  $stmt->execute(array('Panama', 78200, 3652000, 46.70));
  $dbh->commit();

} catch (Exception $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
?>
```

1.5.5 Recuperación de datos

El método *PDOStatement::fetch* permite obtener la siguiente fila de un conjunto de resultados de una consulta. Esta instrucción tiene varios estilos de recuperación,entre ellos:

- PDO::FETCH_NUM: Retorna la siguiente fila como un arreglo indexado por posición.
- PDO::FETCH_ASSOC: Retorna la siguiente fila como un arreglo indexado por el nombre de la columna.
- PDO::FETCH_OBJ: Retorna la siguiente fila como un objeto anónimo con los nombres de las columnas como propiedades.

Si se produce un error, la instrucción *fetch* retornará *FALSE*.

```
<html>
<?php
```

```
try {
  $dbh = new PDO('sqlite:test.db');
} catch (Exception $e) {
  die("Unable to connect: " . $e->getMessage());
}
try {
    $sth = $dbh->prepare("SELECT * FROM countries");
    $sth->execute();
    echo "<table border=1>";
    echo "<tr><th>Country</th><th>Area</th><th>People</th><th>Dens.</th></tr>";
    while ($result = $sth->fetch(PDO::FETCH_ASSOC)) {
        echo "<tr><td>".$result['name']."</td><td>".$result['area'].
            "</td><td>".$result['population']."</td><td>".$result['density'].
            "</td></tr>";
    }
    echo "</table>";
} catch (Exception $e) {
  echo "Failed: " . $e->getMessage();
}
?>
</html>
```

Por su parte la instrucción *PDOStatement::fetchAll* retornará un arreglo conteniendo todos las filas de un conjunto de resultados. El arreglo representa cada columna como un arreglo de valores por columnas o un objeto en donde las propiedades corresponden a los nombres de las columnas. Esta instrucción cuenta con varios modos al igual que la instrucción *fetch*, e inclusive se pueden especificar las columnas que se desean recuperar. Se retorna un arreglo vacío si no existen resultados, o *FALSE* si la consulta falla.

El siguiente ejemplo muestra el uso de la instrucción *fetchAll* , y al mismo tiempo se muestra una forma de recuperar los datos cuando no se conocen de antemano los nombres de las columnas ni la cantidad de ellas.

```
<html>
    <?php
    try {
      $dbh = new PDO('sqlite:test.db');
    } catch (Exception $e) {
      die("Unable to connect: " . $e->getMessage());
    }

    try {
        $sth = $dbh->prepare("SELECT * FROM countries");
        $sth->execute();
        echo "<table border=1><tr>";
        $result = $sth->fetchAll(PDO::FETCH_ASSOC);
        $keys = array_keys($result[0]);
        foreach ($keys as $key)
          echo "<th>".$key."</th>";
        echo "</tr>";
        foreach ($result as $row) {
          echo "<tr>";
          foreach ($keys as $key)
              echo "<td>".$row[$key]."</td>";
          echo "</tr>";
        }
        echo "</table>";
    } catch (Exception $e) {
      echo "Failed: " . $e->getMessage();
    }
```

```
    ?>
</html>
```

1.6 Manipulando XML y JSON

1.6.1 Lectura de datos XML

El método *XMLReader::open*, utilizado como método estático, establece el URL de entrada para el contenido XML que se procesará y *XMLReader::close* cierra dicha entrada. La función *XMLReader::read* mueve al siguiente nodo en el documento. Por su parte la función *XMLReader::next* mueve el cursor al siguiente nodo saltándose todos los subárboles.

Para obtener el valor de un atributo se utiliza la función *XMLReader::getAttribute(name)*. Para obtener el valor (texto) de un nodo se puede utilizar el atributo *XMLReader:value* y para conocer el nombre del elemento se utiliza el atributo *XMLRead:name*. El tipo del nodo se obtiene mediante *XMLReadear::nodeType*. Algunos tipos de nodo son: *XMLReader::ELEMENT*, *XMLReader::TEXT*, *XMLReader::END_ELEMENT*.

Considere el siguiente archivo en formato XML con información sobre varios países:

```
<countries>
  <country name="Belice" area="22966"
           people="334000" density="14.54"/>
  <country name="Costa Rica" area="51100"
           people="4726000" density="92.49"/>
  <country name="El Salvador" area="21041"
           people="6108000" density="290.29"/>
  <country name="Guatemala" area="108894"
           people="15284000" density="140.36"/>
  <country name="Honduras" area="112492"
           people="8447000" density="75.09"/>
  <country name="Nicaragua" area="129494"
           people="6028000" density="46.55"/>
  <country name="Panama" area="78200"
           people="3652000" density="46.70"/>
</countries>
```

El siguiente programa genera una tabla en formato HTML a partir del anterior archivo de datos:

```
<?php

$path = "/Users/xxx/data.xml";

if (!file_exists($path))
    exit("File not found");
$xml = XMLReader::open($path);
echo "<html><body><table border=1>";
echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th></tr>";
while ($xml->read())
if ($xml->nodeType == XMLReader::ELEMENT && $xml->name == 'country') {
    $fields = array();
    $fields[0] = $xml->getAttribute('name');
    $fields[1] = $xml->getAttribute('area');
    $fields[2] = $xml->getAttribute('people');
    $fields[3] = $xml->getAttribute('density');
    echo "<tr><td>".$fields[0]."</td><td>".$fields[1]."</td><td>".
```

```
        $fields[2]."</td><td>".$fields[3]."</td></tr>";
}
echo "</table></body></html>";
$xml->close();

?>
```

1.6.2 Escritura de datos mediante XML

La clase *XMLWriter* crea un objeto de escritura XML, y el método *XMLWriter::openURI(path)* establece el URI de salida para el contenido XML que se generará y *XMLReader::flush* escribe dicha entrada. La función *XMLReader::startDocument* crea el nodo principal de un documento xml, y el método *XMLReader::endDocument* finaliza el documento. Por su parte la función *XMLReader::startElement* crea un elemento XML y el método *XMLReader::endElement* lo finaliza.

Para escribir el valor de un atributo se utiliza la función *XMLReader::writeAttribute(name,value)*. Para obtener el valor (texto) de un nodo se puede utilizar el atributo *XMLReader:value*.

El siguiente programa escribe una serie de datos al archivo XML de países:

```
<?php

$path = "/Users/xxx/data2.xml";

$writer = new XMLWriter();
$writer->openURI($path);
$writer->startDocument('1.0');

$writer->startElement('countries');

$writer->startElement('country');
$writer->writeAttribute('name', 'Belice');
$writer->writeAttribute('area', '22966');
$writer->writeAttribute('people', '334000');
$writer->writeAttribute('density', '14.54');
$writer->endElement();

$writer->startElement('country');
$writer->writeAttribute('name', 'Costa Rica');
$writer->writeAttribute('area', '51100');
$writer->writeAttribute('people', '4726000');
$writer->writeAttribute('density', '92.49');
$writer->endElement();

$writer->endElement();

$writer->endDocument();

$writer->flush();

?>
```

1.6.3 Lectura de datos JSON

El método utilizado por PHP para tratar datos JSON es simplemente convertir hileras de texto (string) en formato JSON a arreglos de PHP. Para ello se utiliza la función *json_decode(string)* la cual recibe dicha hilera de texto y retorna el arreglo.

Por ejemplo considere el siguiente archivo de datos en formato JSON:

```
{"countries": [{"name":"Belice","area":"22966","people":"334000","density":"14.54"},
   {"name":"Costa Rica","area":"51100","people":"4726000","density":"92.49"},
   {"name":"El Salvador","area":"21041","people":"6108000","density":"290.29"},
   {"name":"Guatemala","area":"108894","people":"15284000", "density":"140.36"},
   {"name":"Honduras","area":"112492","people":"8447000", "density":"75.09"},
   {"name":"Nicaragua","area":"129494","people":"6028000","density":"46.55"},
   {"name":"Panama", "area":"78200","people":"3652000","density":"46.70"}
 ]}
```

Un programa para leer el anterior archivo y generar una tabla HTML con dicha información, sería el siguiente:

```
<?php

$path = "/Users/xxx/data.json";

if (!file_exists($path))
    exit("File not found");

$data = file_get_contents($path);
$json = json_decode($data, true);

echo "<html><body><table border=1>";
echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th></tr>";
foreach ($json['countries'] as $row) {
    echo "<tr><td>".$row['name']."</td><td>".$row['area']."</td><td>".
         $row['people']."</td><td>".$row['density']."</td></tr>";
}
echo "</table></body></html>";

?>
```

1.6.4 Escritura de datos JSON

De igual forma para escribir datos en formato JSON, PHP utiliza la función *json_encode(arras)* la cual recibe un arreglo de PHP y retorna una hilera de texto en formato JSON.

El siguiente programa genera parte del archivo de datos de países en formato JSON:

```
<?php

$path = "/Users/xxx/data2.json";
$file = fopen($path, "w");

$countries = array(
    array("name"=>"Belice","area"=>"22966","people"=>"334000","density"=>"14.54"),
    array("name"=>"Costa Rica","area"=>"51100","people"=>"4726000","density"=>"92.49")
);

$json = json_encode($countries);
```

```
fwrite($file, $json);
fclose($file);

?>
```

1.7 Generación de imágenes

PHP permite la generación de imágenes mediante una serie de primitivas de dibujo. El primer paso para realizar esto consiste en definir el tamaño de la imagen mediante la instrucción *imagecreate(width,height)*. Luego se aplican sobre dicha imagen las primitivas de dibujo deseadas, al final se debe usar una instrucción para convertir la imagen en algún formato conocido como: *imagepng(image)*, *imagegif(image)* ó *imagejpeg(image)*. Posteriormente, se puede destruir la imagen en memoria utilizando la instrucción *imagedestroy(image)*. Si la imagen se desea desplegar en el navegador es necesario enviar el encabezado adecuado, por ejemplo:

```
header( "Content-type: image/png" )
```

1.7.1 Definición de colores

Para definir los colores que se utilizarán con las diferentes primitivas de dibujo es necesario utilizar la instrucción *imagecolorallocate(image, red, green, blue)*. Los rangos de los valores de rojo, verde y azul deben estar entre 0 y 255. Estos valores también pueden ser especificados en hexadecimal, por ejemplo: 0xFF

Es importante indicar que la primer llamada que se realice a esta función definirá automáticamente el color de fondo (background) de la imagen, por ejemplo:

```
<?php
     $myImage = imagecreate( 200, 100 );
     $myGray = imagecolorallocate( $myImage, 204, 204, 204 );
     header( "Content-type: image/png" );
     imagepng( $myImage );
     imagedestroy( $myImage );
?>
```

1.7.2 Primitivas de dibujo

Se cuenta con diferentes primitivas de dibujo que permiten crear gráficos sobre la imagen. Existen primitivas para dibujar el borde de diferentes geometrías:

Figura	Instrucción
Línea	imageline(image, posX1, posY1, posX2, posY2, color)
Rectángulo	imagerectangle(image, minX, minY, maxX, maxY, color)
Elipse	imageellipse(image, posX, posY, width, height, color)
Polígono	imagepolygon(image, array, num_points, color)

Para dibujar geometrías rellenas se deben usar las instrucciones:

Figura	Instrucción
Rectángulo	imagefilledrectangle(image, minX, minY, maxX, maxY, color)
Elipse	imagefilledellipse(image, posX, posY, width, height, color)
Polígono	imagefilledpolygon(image, array, num_points, color)

```
<?php
     $myImage = imagecreate( 200, 100 );
     $myGray = imagecolorallocate( $myImage, 204, 204, 204 );
     $myBlack = imagecolorallocate( $myImage, 0, 0, 0 );
     imageline( $myImage, 15, 35, 120, 60, $myBlack );
     imagerectangle( $myImage, 15, 35, 120, 60, $myBlack );
     imageellipse( $myImage, 90, 60, 160, 50, $myBlack );
     $myPoints = array( 20, 20, 185, 55, 70, 80 );
     imagepolygon( $myImage, $myPoints, 3, $myBlack );
     header( "Content-type: image/png" );
     imagepng( $myImage );
     imagedestroy( $myImage );
?>
```

1.7.3 Manejo de texto

Es posible incluir texto en las imágenes para lo cual se puede utilizar la instrucción *imagestring(image, font, posX, posY, string, color)*. El tipo de fuente (font) puede ser un número entre 1 y 5 para los tipos de fuente predefinidos. Es posible cargar nuevas fuentes de letra desde archivos en formato *gdf* mediante la instrucción *imageloadfont(filename)*. También se puede obtener el ancho y tamaño del tipo de letra actual mediante las instrucciones *imagefontwidh()* e *imagefontheight()*.

```
<?php
     $textImage = imagecreate( 200, 100 );
     $white = imagecolorallocate( $textImage, 255, 255, 255 );
     $black = imagecolorallocate( $textImage, 0, 0, 0 );
     $yOffset = 0;
for ( $i = 1; $i <= 5; $i++ ) {
imagestring( $textImage, $i, 5, $yOffset, "This is system font $i", $black
     );
       $yOffset += imagefontheight( $i );
}
     header("Content-type: image/png");
     imagepng( $textImage );
     imagedestroy( $textImage );
?>
```

En ocasiones es conveniente utilizar tipos de letra *TrueType* que mejoren la apariencia de la imagen. Para cargar archivos de este tipo se utiliza la instrucción *imagefttext(image, size, angle, posX, posY, color, filename, string)*

```
<?php
      $textImage = imagecreate( 200, 120 );
      $white = imagecolorallocate( $textImage, 255, 255, 255 );
      $black = imagecolorallocate( $textImage, 0, 0, 0 );
      imagefttext( $textImage, 16, 0, 10, 50, $black, "fonts/truetype/
      ttf-bitstream-vera/Vera.ttf", "Vera, 16 pixels" );
      header( "Content-type: image/png" );
      imagepng( $textImage );
      imagedestroy( $textImage );
?>
```

1.8 Plantillas

Mustache es un motor de plantillas para PHP, es decir, separa el código PHP, como lógica de negocios, del código HTML, como lógica de presentación, y genera contenidos web mediante la colocación de etiquetas *mustache* en un documento. *Mustache* está disponible en muchos lenguajes de programación y es consistente entre plataformas.

La versión para PHP puede ser descargada desde https://github.com/bobthecow/mustache.php. Unicamente es necesario copiar, en el directorio en donde se correrá el programa, el directorio *src/mustache*.

1.8.1 Uso básico

Un programa PHP debe realizar la inicialización del motor *mustache* mediante el *autoloader* que viene incluido con el código. En dicha inicialización se puede indicar la extensión y ubicación de los archivos de plantillas, por ejemplo */views*. Dentro del programa la plantilla se carga mediante el método *loadTemplate* y luego se puede *aplicar* mediante la función *render*. Note que a esta última función se deben pasar como parámetros, mediante un arreglo, el conjunto de datos que serán utilizados por la plantilla.

```
<?php

require 'Mustache/Autoloader.php';
Mustache_Autoloader::register();

$mustache = new Mustache_Engine(array(
    'loader' => new Mustache_Loader_FilesystemLoader(
                        dirname(__FILE__) . '/views',
                        array('extension' => '.tpl')),
));

$path = "data.json";
if (!file_exists($path))
    exit("File not found");

$data = file_get_contents($path);
$json = json_decode($data, true);

$tpl = $mustache->loadTemplate('Example9_1.tpl');
echo $tpl->render(array('countries' => $json['countries']));

?>
```

El mecanismo de marcado (etiquetas) de *mustache* utiliza pares de «corchetes» para definir el inicio y final de cada bloque de elementos. Por ejemplo, para el programa anterior la plantilla se vería de la siguiente forma. Recuerde que esta plantilla tiene como nombre *Example9_1.tpl* y se encuentra bajo el directorio */views*

```
<html><body><table border=1>
<tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th></tr>
{{#countries}}
    <tr><td>{{name}}</td><td>{{area}}</td><td>
         {{people}}</td><td>{{density}}</td></tr>
{{/countries}}
</table></body></html>
```

Es importante indicar que el lenguaje de marcado de *mustache* es consistente entre las distintas implementaciones de la librería, en diferentes lenguajes de programación. Por lo tanto no se puede incluir código PHP en una plantilla de *mustache*, pero dicha plantilla puede ser «transportada» a otro ambiente de programación como: javascript, python, Java, C++ ó ASP; y seguirá funcionando de la misma forma.

1.8.2 Condicionales

Es posible indicar que cierto contenido debe aparecer en la salida bajo ciertas condiciones y otro no. Para eso se utiliza combinaciones de etiquetas con «#» y «^». Por ejemplo considere el siguiente programa que utiliza algunos datos de países, pero algunos están incompletos.

```
<?php

require 'Mustache/Autoloader.php';
Mustache_Autoloader::register();

$mustache = new Mustache_Engine(array(
    'loader' => new Mustache_Loader_FilesystemLoader(
                        dirname(__FILE__) . '/views',
                        array('extension' => '.tpl')),
));

$countries = array(
    array("name"=>"Belice","area"=>"22966","people"=>"334000","density"=>"14.54"),
    array("area"=>"33444","people"=>"3434340","density"=>"0"),
    array("name"=>"Costa Rica","area"=>"51100","people"=>"4726000","density"=>"92.49
↪"),
    array("area"=>"229656","people"=>"99800","density"=>"0"),
);

$tpl = $mustache->loadTemplate('Example9_2.tpl');
echo $tpl->render(array('countries' => $countries));

?>
```

La plantilla asociada, llamada *Example9_2.tpl*, mostraría una etiqueta «desconocido» en aquellos registros que no aparezca su nombre, tal como se muestra a continuación:

```
<html><body><table border=1>
<tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th></tr>
{{#countries}}
  {{#name}}
    <tr><td>{{name}}</td><td>{{area}}</td><td>
         {{people}}</td><td>{{density}}</td></tr>
  {{/name}}
  {{^name}}
    <tr><td>Desconocido</td><td>{{area}}</td><td>
         {{people}}</td><td>{{density}}</td></tr>
  {{/name}}
{{/countries}}
</table></body></html>
```

1.9 Servicios Web tipo RESTfull

La librería *ToroPHP* permite la creación de servicios web tipo RESTfull de forma sencilla y eficiente en PHP. Esta se encuentra disponible en http://github.com/anandkunal/ToroPHP. Una vez que se descarga el archivo *Toro.php* éste puede ser probado utilizando un pequeño programa de ejemplo, llamado *prueba.php*:

```
<?php
```

```
require("Toro.php");

class HelloHandler {
    function get() {
        echo "Hello, world";
    }
}

Toro::serve(array(
    "/" => "HelloHandler",
));

?>
```

Dicho programa se puede ejecutar mediante el url *http://localhost/webservice/prueba.php/*

1.9.1 Consulta de datos

En el protocolo *RESTfull* se utiliza el método *GET* para recuperar información. Para identificar el tipo de elemento a consultar se puede utilizar un «subdirectorio» en el URL.

Por ejemplo, considere el siguiente programa que recupera todos datos de los países desde la base de datos y los presenta en formato JSON. Para invocarlo se debe usar un URL como *http://localhost/prueba.php/country*.

```
<?php
require("Toro.php");

class DBHandler {
    function get() {
        try {
          $dbh = new PDO('sqlite:test.db');
        } catch (Exception $e) {
          die("Unable to connect: " . $e->getMessage());
        }
        try {
            $stmt = $dbh->prepare("SELECT * FROM countries");
            $stmt->execute();

            $data = Array();
            while ($result = $stmt->fetch(PDO::FETCH_ASSOC)) {
                $data[] = $result;
            }
            echo json_encode($data);
        } catch (Exception $e) {
          echo "Failed: " . $e->getMessage();
        }
    }
}

Toro::serve(array(
    "/country" => "DBHandler",
));
?>
```

Si se desea invocar a este servicio web desde otro programa PHP basta con definir el contenido a partir de la dirección URL del servicio, por ejemplo:

```
<?php

    $path = "http://localhost/prueba.php/country";

    $data = file_get_contents($path);
    $json = json_decode($data, true);

    echo "<html><body><table border=1>";
    echo "<tr><th>Country</th><th>Area</th><th>Population</th><th>Density</th></tr>";
    foreach ($json as $row) {
        echo "<tr><td>".$row['name']."</td><td>".$row['area']."</td><td>".
             $row['population']."</td><td>".$row['density']."</td></tr>";
    }
    echo "</table></body></html>";

?>
```

Paso de parámetros

En los servicios web tipo RESTfull, y en ToroPHP en particular, los parámetros de la consulta son pasados también como se fueran subdirectorios y no mediante el símbolo ?. Por ejemplo, una llamada que normalmente se realizaría de la siguiente forma:

```
http://localhost/prueba.php?table=country&name=Nicaragua
```

se realizaría de la siguiente forma utilizando un servicio RESTfull:

```
http://localhost/prueba.php/country/Nicaragua
```

note que en este caso el nombre de los parámetros no aparecen y ellos deben ser identificados de forma implícita por su posición en la solicitud.

Por ejemplo, considere ahora una modificación al programa anterior que recuperará los datos de un país en la base de datos y los presenta en formato JSON. Para invocarlo se debe usar un URL como *http://localhost/prueba.php/country/Nicaragua*.

```
<?php
    require("Toro.php");

    class DBHandler {

        function get($name=null) {
            try {
              $dbh = new PDO('sqlite:test.db');
            } catch (Exception $e) {
              die("Unable to connect: " . $e->getMessage());
            }
            try {
                if ($name!=null) {
                    $stmt = $dbh->prepare("SELECT * FROM countries WHERE name = :name
↪");
                    $stmt->bindParam(':name', $name, PDO::PARAM_STR);
                } else {
                    $stmt = $dbh->prepare("SELECT * FROM countries");
                }
                $stmt->execute();
```

```
                $data = Array();
                while ($result = $stmt->fetch(PDO::FETCH_ASSOC)) {
                    $data[] = $result;
                }
                echo json_encode($data);
            } catch (Exception $e) {
              echo "Failed: " . $e->getMessage();
            }
        }
    }

    Toro::serve(array(
        "/country" => "DBHandler",
        "/country/:alpha" => "DBHandler",
    ));
?>
```

Note que existen tres tipos de parámetros que reconoce ToroPHP: number, alpha y string; o bien se puede utilizar una expresión regular como: ([0-9]+), ([a-zA-Z0-9-_]+) ó ([a-zA-Z]+). Note que pueden ser utilizados múltiples parámetros en la solicitud, y estos serán pasados en el orden en que aparecen al método *get* de la clase utilizada como manejador (handler).

1.9.2 Envío de datos

Para enviar información a un servicio *RESTfull* se utiliza el método *POST* ó *PUT*. Generalmente el método *PUT* se utiliza para crear un elemento, y el método *POST* para modificar los datos de un elemento existente. Una nueva modificación al servicio web incorpora la capacidad de modificar los datos de un registro, tal como se muestra a continuación:

```
<?php
    require("Toro.php");

    class DBHandler {

        function get($name=null) {
            // como en el ejemplo anterior
        }

        function post($name=null) {
            try {
              $dbh = new PDO('sqlite:test.db');
            } catch (Exception $e) {
              die("Unable to connect: " . $e->getMessage());
            }
            try {
              $area = $_POST['area'];
              $population = $_POST['population'];
              $density = $_POST['density'];
              echo $area;

              $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

              $stmt = $dbh->prepare("UPDATE countries SET area=:area,
                                     population=:population, density=:density
                                     WHERE name = :name");
```

```
                $stmt->bindParam(':area', $area);
                $stmt->bindParam(':population', $population);
                $stmt->bindParam(':density', $density);

                $dbh->beginTransaction();
                $stmt->execute();
                $dbh->commit();
                echo 'Successfull';
            } catch (Exception $e) {
                $dbh->rollBack();
                echo "Failed: " . $e->getMessage();
            }
        }
    }

    Toro::serve(array(
        "/country" => "DBHandler",
        "/country/:alpha" => "DBHandler",
    ));
?>
```

Sin embargo para invocar esta función del servicio web, desde otro programa PHP, es necesario utilizar un *stream PHP* tal como se muestra a continuación:

```
<?php

$name='Nicaragua';
$url = 'http://localhost/prueba.php/country/'.$name;

$data = array('name'=>'Nicaragua','area'=>'129000',
              'population'=>'6548000','density' => '46.55');
$options = array(
        'http' => array(
        'header'  => "Content-type: application/x-www-form-urlencoded\r\n",
        'method'  => 'POST',
        'content' => http_build_query($data),
    )
);

$context  = stream_context_create($options);
$result = file_get_contents($url, false, $context);

echo $result;
?>
```