

# Funciones y objetos en PHP

Los requisitos básicos de cualquier lenguaje de programación son disponer de un lugar para almacenar datos, un medio con el que dirigir el flujo del programa y algunos elementos como las evaluaciones de expresiones, el manejo de archivos y la salida de texto. PHP tiene todo esto, además de herramientas como `else` y `elseif` para hacernos la vida más fácil. Pero incluso si dispones de todo esto en tu kit de herramientas, la programación puede resultar chapucera y tediosa, especialmente si tienes que reescribir partes de documentos muy similares cada vez que los necesites.

Ahí es donde entran en juego las funciones y los objetos. Como puedes suponer, una *función* es un conjunto de declaraciones que realizan un cometido específico y, opcionalmente, proporcionan un valor. Puedes extraer una sección de código que has usado más de una vez, colocarla en una función y llamar a la función por su nombre cuando necesites aplicar esa sección de código.

Las funciones tienen muchas ventajas sobre el código en línea contiguo. Por ejemplo:

- Supone escribir menos.
- Disminuye el número de errores de sintaxis y otros errores de programación
- Se reduce el tiempo de carga de los archivos de programas.
- Se reduce el tiempo de ejecución, ya que cada función se compila solo una vez, sin importar la frecuencia con la que la invocamos.
- Aceptan argumentos y, por lo tanto, se pueden utilizar tanto para casos generales como específicos.

Los objetos llevan este concepto un paso más allá. Un *objeto* incorpora una o más funciones, así como los datos que utilizan, a una sola estructura llamada *clase*.

En este capítulo aprenderás todo sobre el uso de las funciones, desde definir las y llamarlas hasta pasar argumentos de un lado a otro. Una vez hayas asimilado estos conocimientos, empezarás a crear funciones y a utilizarlas en tus objetos (y se hará referencia a ellas como *métodos*).



Ahora no es habitual (y definitivamente no se recomienda) usar cualquier versión de PHP inferior a 5.4. Por lo tanto, en este capítulo se supone que esta versión es la versión mínima con la que estarás trabajando. Generalmente recomendaría la versión 5.6, o las nuevas versiones 7.0 o 7.1 (no hay versión 6). Puedes seleccionar cualquiera de ellas en el panel de control AMPPS, tal y como se describe en el Capítulo 2.

# Funciones en PHP

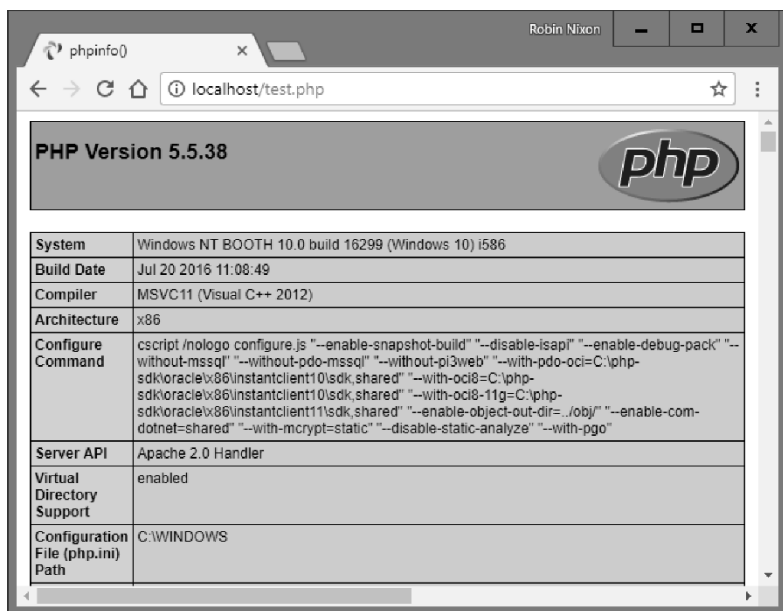
PHP viene con cientos de funciones preconfiguradas e integradas, lo que lo convierte en un lenguaje muy rico. Para usar una función, tienes que llamarla por su nombre. Por ejemplo, puedes ver cómo opera la función `date` escribiendo:

```
echo date("l"); // Displays the day of the week
```

Los paréntesis le dicen a PHP que te estás refiriendo a una función. De lo contrario, pensaría que te refieres a una constante.

Las funciones pueden admitir cualquier número de argumentos, y ninguno. Por ejemplo, `phpinfo`, como se muestra a continuación, presenta mucha información sobre la instalación en uso de PHP y no necesita explicación. El resultado de llamar a esta función se puede ver en la Figura 5-1.

```
phpinfo();
```



System	Windows NT BOOTH 10.0 build 16299 (Windows 10) i586
Build Date	Jul 20 2016 11:08:49
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscrip /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--with-out-pl3web" "--with-pdo-oci=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\x86\instantclient11\sdk,shared" "--enable-object-out-dir=.\/obj" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS

**Figura 5-1.** Salida de la función integrada `phpinfo` de PHP



La función `phpinfo` es extremadamente útil para obtener información sobre la instalación que tengas funcionando de PHP, pero esa información también podría ser muy útil para potenciales hackers. Por lo tanto, nunca dejes una llamada a esta función en cualquier código preparado para la web.

Algunas de las funciones integradas que usan uno o más argumentos aparecen en el Ejemplo 5-1.

### Ejemplo 5-1. Tres funciones tipo cadena

```
<?php
echo strrev(" .dlrow olleH");    // Reverse string
echo str_repeat("Hip ", 2);      // Repeat string
echo strtoupper("hooray!");      // String to uppercase
?>
```

Este ejemplo utiliza tres funciones de cadena para proporcionar el siguiente texto:

**Hello world. Hip Hip HOORAY!**

Como puedes ver, la función `strrev` invierte el orden de los caracteres en la cadena, `str_repeat` repite la cadena "Hip" dos veces (como requiere el segundo argumento) y `strtoupper` convierte "hooray!" en mayúsculas.

## Definición de función

La sintaxis general de una función es la siguiente:

```
function function_name([parameter [, ...]])
{
    // Statements
}
```

La primera línea indica lo siguiente:

- La definición comienza con la palabra `function`.
- A continuación aparece un nombre que debe comenzar con una letra o guion bajo, seguido de cualquier número de letras, números o guiones bajos.
- Los paréntesis son obligatorios.
- Son opcionales uno o más parámetros separados por comas (como se indica entre corchetes).

Los nombres de las funciones no distinguen entre mayúsculas y minúsculas, por lo que todas las cadenas que se muestran a continuación pueden referirse a la función imprimir: `PRINT`, `print` y `PrInT`.

La llave de apertura inicia las declaraciones que se ejecutarán cuando llames a la función, que acaba con una llave de cierre. Estas declaraciones pueden incluir una o más declaraciones `return`, que obligan a la función a detener la ejecución y volver al código desde el que se la ha llamado. Si se asigna un valor a la declaración `return`, el código de llamada puede recuperarlo, como veremos a continuación.

## Devolución de un valor

Echemos un vistazo a una simple función para convertir el nombre completo de una persona a letras minúsculas y, a continuación, escribir en mayúsculas la primera letra de cada parte del nombre.

## Aprender PHP, MySQL y JavaScript

Ya hemos visto un ejemplo de la función `strtoupper` integrada en PHP en el Ejemplo 5-1. En nuestra función usaremos su equivalente, `strtolower`:

```
$lowered = strtolower("aNY # of Letters and Punctuation you
WANT");
echo $lowered;
```

El resultado de este experimento es el siguiente:

```
any # of letters and punctuation you want
```

Sin embargo, no queremos que todos los nombres estén en minúsculas; queremos que la primera letra de cada parte del nombre esté en mayúsculas. (No vamos a tratar casos sutiles como Mary-Ann o Jo-En-Lai para este ejemplo). Afortunadamente, PHP también proporciona una función `ucfirst` que establece el primer carácter de una cadena en mayúsculas:

```
$ucfixed = ucfirst("any # of letters and punctuation you want");
echo $ucfixed;
```

La salida es la siguiente:

```
Any # of letters and punctuation you want
```

Ahora podemos diseñar nuestra primera parte del programa: para poner una palabra con su letra inicial en mayúsculas; lo primero que hacemos es llamar a `strtolower` para que actúe sobre la cadena y, luego, a `ucfirst`. La manera de hacer esto es anidar una llamada a `strtolower` dentro de `ucfirst`. Veamos por qué, porque es importante entender el orden en el que se evalúa el código.

Digamos que solo haces una llamada a la función `print`:

```
print(5-8);
```

En primer lugar se evalúa la expresión `5-8`, y la salida es `-3`. (Como has visto en el capítulo anterior, PHP convierte el resultado a una cadena para mostrarlo). Si la expresión contiene una función, esa función también se evalúa en primer lugar:

```
print(abs(5-8));
```

PHP hace varias cosas al ejecutar esa breve declaración:

1. Evalúa `5-8` y da `-3`.
2. Utiliza la función `abs` para convertir `-3` en `3`.
3. Convierte el resultado a una cadena y la salida utiliza la función `print`.

Todo funciona porque PHP evalúa cada elemento de dentro hacia fuera. Se utiliza el mismo procedimiento cuando hacemos una llamada a lo siguiente:

```
ucfirst(strtolower("aNY # of Letters and Punctuation you WANT"))
```

PHP pasa nuestra cadena a `strtolower` y luego a `ucfirst`, y produce (como ya hemos visto cuando jugamos con las funciones por separado) lo siguiente:

```
Any # of letters and punctuation you want
```

Ahora vamos a definir una función (mostrada en el Ejemplo 5-2) que emplea tres nombres y convierte cada uno en minúsculas, con la letra inicial en mayúscula.

### Ejemplo 5-2. Escritura correcta de un nombre completo

```
<?php
    echo fix_names("WILLIAM", "henry", "gatES");

    function fix_names($n1, $n2, $n3)
    {
        $n1 = ucfirst(strtolower($n1));
        $n2 = ucfirst(strtolower($n2));
        $n3 = ucfirst(strtolower($n3));

        return $n1 . " " . $n2 . " " . $n3;
    }
?>
```

Es posible que tengas que escribir este tipo de código, porque los usuarios a menudo dejan la tecla Caps Lock (bloqueo de mayúsculas) activada, accidentalmente insertan mayúsculas en los lugares equivocados, e incluso olvidan las mayúsculas por completo. El resultado de este ejemplo se muestra aquí:

**William Henry Gates**

## Devolución de una matriz

Acabamos de ver una función que devuelve un solo valor. También hay formas de obtener múltiples valores de una función.

El primer método es devolverlos en una matriz. Como se vio en el Capítulo 3, una matriz es como un montón de variables pegadas unas a otras formando una fila. El Ejemplo 5-3 muestra cómo puedes usar una matriz para devolver los valores de la función.

### Ejemplo 5-3. Devolución de varios valores en una matriz

```
<?php
    $names = fix_names("WILLIAM", "henry", "gatES");
    echo $names[0] . " " . $names[1] . " " . $names[2];

    function fix_names($n1, $n2, $n3)
    {
        $n1 = ucfirst(strtolower($n1));
        $n2 = ucfirst(strtolower($n2));
        $n3 = ucfirst(strtolower($n3));

        return array($n1, $n2, $n3);
    }
?>
```

Este método tiene la ventaja de mantener los tres nombres separados, en lugar de concatenarlos formando una sola cadena, por lo que puedes hacer referencia a cualquier

usuario simplemente por su nombre o apellido sin estar obligado a extraer ninguno de los dos nombres de la cadena devuelta.

### Paso de argumentos por referencia

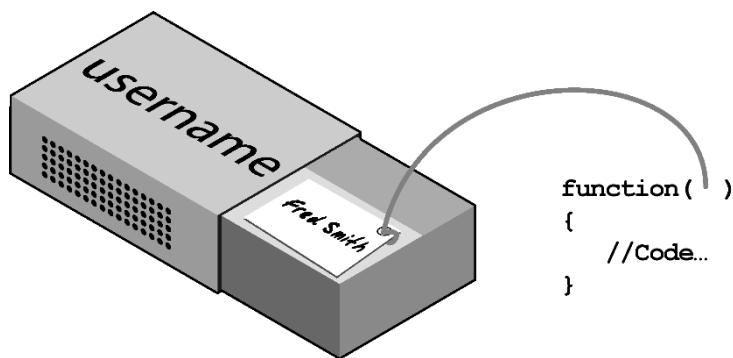
En las versiones de PHP anteriores a la 5.3, se podía anteponer a una variable el símbolo `&` en el momento de llamar a una función (por ejemplo, `increment (&$myvar) ;`) para decirle al analizador que pasara una referencia de la variable, no del valor de la variable. Esto permitía a una función acceder a la variable (y permitía que se volvieran a escribir diferentes valores en ella).



La llamada al paso de argumentos por referencia se quedó obsoleta en PHP 5.3 y se eliminó en PHP 5.4. Por lo tanto, no debes usar esta característica más que en sitios web heredados, e incluso en esos casos se recomienda reescribir el código de paso por referencia, porque en las versiones más recientes de PHP se detendrá y producirá un error fatal.

Sin embargo, *dentro* de la definición de función, puedes continuar accediendo a los argumentos por referencia. Este concepto puede ser difícil de entender, así que volvamos a la metáfora de la caja de cerillas del Capítulo 3.

Imagínate que, en lugar de sacar un trozo de papel de una caja de cerillas, leerlo, copiar su contenido en otro trozo de papel, poner el original de nuevo en la caja, y pasar la copia a una función (¡uf!), puedes simplemente atar el extremo de un trozo de hilo al original y pasar el otro extremo a la función (ver la Figura 5-2).



**Figura 5-2.** Imaginar una referencia como un hilo unido a una variable

Ahora la función puede seguir el hilo para encontrar los datos a los que debe acceder. Esto evita la sobrecarga de crear una copia de la variable solo para que la utilice la función. Además, la función ahora puede modificar el valor de la variable.

Esto significa que puedes reescribir el Ejemplo 5-3 para pasar referencias a todos los parámetros, y entonces la función puede modificarlos directamente (ver Ejemplo 5-4).

### Ejemplo 5-4. Paso de valores a una función por referencia

```
<?php
    $a1 = "WILLIAM";
    $a2 = "henry";
    $a3 = "gatES";

    echo $a1 . " " . $a2 . " " . $a3 . "<br>"; fix_names($a1, $a2,
$a3);
    echo $a1 . " " . $a2 . " " . $a3;

    function fix_names(&$n1, &$n2, &$n3)
    {
        $n1 = ucfirst(strtolower($n1));
        $n2 = ucfirst(strtolower($n2));
        $n3 = ucfirst(strtolower($n3));
    }
?>
```

En lugar de pasar cadenas directamente a la función, primero se asignan a variables y se imprimen para ver sus valores "antes". Luego llamas a la función como antes, pero dentro de la definición de la función se coloca un símbolo & delante de cada parámetro que se debe pasar por referencia.

Ahora las variables \$n1, \$n2 y \$n3 se adjuntan a "hilos" que conducen a los valores \$a1, \$a2 y \$a3. En otras palabras, hay un grupo de valores, pero se permite el acceso a estos valores a dos conjuntos de nombres de variables.

Por lo tanto, la función `fix_names` solo tiene que asignar nuevos valores a \$n1, \$n2 y \$n3 para actualizar los valores de \$a1, \$a2 y \$a3. La salida de este código es:

```
WILLIAM henry gatES
William Henry Gates
```

Como puedes ver, ambas declaraciones de `echo` usan solo los valores \$a1, \$a2 y \$a3.

## Devolución en variables globales

La mejor manera de que una función tenga acceso a una variable creada externamente es declarando que tiene acceso global desde dentro de la función. La palabra clave `global` seguida del nombre de la variable le da a cualquier parte del código acceso completo a ella (ver el Ejemplo 5-5).

### Ejemplo 5-5. Devolución de valores en variables globales

```
<?php
    $a1 = "WILLIAM";
    $a2 = "henry";
    $a3 = "gatES";

    echo $a1 . " " . $a2 . " " . $a3 . "<br>"; fix_names();
    echo $a1 . " " . $a2 . " " . $a3;
```

```
function fix_names()  
{  
    global $a1; $a1 = ucfirst(strtolower($a1));  
    global $a2; $a2 = ucfirst(strtolower($a2));  
    global $a3; $a3 = ucfirst(strtolower($a3));  
}  
?>
```

Ahora no tienes que pasar parámetros a la función, y esta no tiene que aceptarlos. Una vez declaradas, estas variables conservan el acceso global y están disponibles para el resto del programa, incluidas sus funciones.

## Recapitulación sobre el ámbito de aplicación de las variables

Un recordatorio rápido de lo que sabes del Capítulo 3:

- Las *variables locales* son accesibles solo desde la parte del código en el que las defines. Si están fuera de una función se puede acceder a ellas mediante cualquier código externo a las funciones, clases, etc. Si una variable está dentro de una función, solamente esa función puede acceder a la variable, y su valor se pierde cuando la función deja de ejecutarse.
- Las *variables globales* son accesibles desde cualquier parte del código.
- Las *variables estáticas* son accesibles solamente dentro de la función que las ha declarado, pero mantienen su valor después de varias llamadas a la función.

## Inclusión y requisición de archivos

A medida que progreses en el uso de la programación PHP, es probable que comiences a crear una biblioteca de funciones que piensas que necesitarás de nuevo. También es probable que empieces a usar bibliotecas creadas por otros programadores.

No necesitas copiar y pegar estas funciones en tu código. Puedes guardarlas en archivos separados y usar los comandos que existen para extraerlas: `include` y `require`.

### La declaración `include`

Si utilizas `include`, puedes decirle a PHP que obtenga un archivo en particular y cargue todo su contenido. Es como si hubieras pegado el archivo a incluir en el archivo en curso, en el punto en el que lo has insertado. El Ejemplo 5-6 muestra cómo se incluiría un archivo llamado *library.php*.

#### Ejemplo 5-6. Inclusión de un archivo PHP

```
<?php  
    include "library.php";  
    // Your code goes here  
?>
```



### Utilización de `include_once`

Cada vez que emites la directiva `include`, vuelve a incluir el archivo solicitado, incluso si ya lo has insertado. Por ejemplo, supongamos que *library.php* contiene muchas cosas útiles por lo que lo incluyes en tu archivo, pero también incluyes otra biblioteca que incluye *library.php*. A través del anidamiento, has incluido sin querer *library.php* dos veces. Esto producirá mensajes de error, porque estás intentando definir la misma constante o función varias veces. Por lo tanto, deberías utilizar `include_once` en lugar de `include` (ver Ejemplo 5-7).

**Ejemplo 5-7.** Inclusión de un archivo PHP solamente una vez

```
<?php
    include_once "library.php";

    // Your code goes here
?>
```

Así se ignorará cualquier otro intento de incluir el mismo archivo (con `include` o `include_once`). Para determinar si se ha ejecutado la solicitud del archivo, se comprueba la ruta de archivo absoluta después de resolver todas las rutas relativas, y el archivo se debe encontrar en la ruta que hayas especificado en `include`.



En general, probablemente es mejor atenerse a `include_once` e ignorar la declaración básica `include`. De esa manera, nunca tendrás el problema de que los archivos se incluyan varias veces.

### Utilización de `require` y `require_once`

Un problema potencial con `include` e `include_once` es que PHP solo intentará incluir el archivo solicitado. La ejecución del programa continúa incluso si no se encuentra el archivo.

Cuando sea absolutamente imprescindible incluir un archivo, usa `require`. Por las mismas razones que expuse para usar `include_once`, recomiendo que generalmente te quedes con `require_once` cuando necesites requerir un archivo (ver el Ejemplo 5-8).

**Ejemplo 5-8.** Requisición de un archivo PHP solo una vez

```
<?php
    require_once "library.php";

    // Your code goes here
?>
```

# Compatibilidad de las versiones PHP

PHP está en un proceso continuo de desarrollo, y hay múltiples versiones. Si necesitas verificar si una función determinada está disponible para tu código, puedes utilizar la función `function_exists`, que verifica todas las funciones predefinidas y las creadas por el usuario.

El Ejemplo 5-9 comprueba `array_combine`, una función específica de la versión 5 de PHP.

### Ejemplo 5-9. Verificación de la existencia de una función

```
<?php
    if (function_exists("array_combine"))
    {
        echo "Function exists";
    }
    else
    {
        echo "Function does not exist - better write our own";
    }
?>
```

Si usas un código como este, puedes aprovechar las características de las últimas versiones de PHP y aun así puedes tener tu código ejecutándose en versiones anteriores, siempre y cuando repliques cualquier característica que falte. Las funciones pueden ser más lentas que las integradas, pero al menos tu código será mucho más portátil.

También puedes usar la función `phpversion` para determinar en qué versión de PHP se ejecuta tu código. El resultado devuelto será similar al siguiente, en función de la versión:

5.5.38

# Objetos en PHP

De la misma manera que las funciones representan una gran mejora en la capacidad de la programación en relación con los primeros días de la informática, en los que a veces el mejor programa de navegación disponible era una sentencia `GOTO` o `GOSUB` muy básicas, la *programación orientada a objetos* (POO) lleva el uso de funciones a un nivel completamente nuevo.

Una vez que te acostumbras a comprimir bits de código reutilizables en funciones, no supone un gran salto considerar la agrupación de funciones y sus datos en objetos.

Consideremos un sitio de redes sociales formado por muchas partes. Una de ellas gestiona las funciones de usuario, es decir, código para permitir que los nuevos usuarios se registren y los usuarios existentes modifiquen sus detalles. En PHP estándar, puedes crear algunas funciones para su gestión e integrar algunas llamadas a la base de datos MySQL para hacer un seguimiento de todos los usuarios.

Imagina lo fácil que sería crear un objeto para representar a un usuario. Para hacer esto, podrías crear una clase, tal vez llamada `User`, que contenga los archivos necesarios para la gestión de los usuarios y todas las variables necesarias para la manipulación de los datos dentro de la clase. Entonces, siempre que necesites gestionar los datos de un usuario, podrás simplemente crear un nuevo objeto con la clase `User`.

Podrías tratar este nuevo objeto como si fuera un usuario real. Por ejemplo, podrías darle al objeto un nombre, una contraseña y una dirección de correo electrónico; preguntar si dicho usuario ya existe y, si no, hacer que se cree un nuevo usuario con esos atributos. Podrías incluso tener un objeto de mensajería instantánea, o uno para administrar si dos usuarios son amigos.

### Terminología

Al crear un programa para utilizar objetos, necesitas diseñar una combinación de datos y código llamada *clase*. A cada nuevo objeto basado en esta clase se le llama *instancia* (u *ocurrencia*) de esa clase.

A los datos asociados a un objeto se los denomina *propiedades*; las funciones que utiliza se denominan *métodos*. Al definir una clase, debes proporcionar los nombres de sus propiedades y el código para sus métodos. La Figura 5-3 es una metáfora de la gramola para representar un objeto. Piensa en los CD que tiene en el carrusel como sus propiedades; el método para reproducirlos son los botones del panel frontal. También hay una ranura para insertar monedas (método utilizado para activar el objeto) y el lector de discos láser (el método utilizado para recuperar la música, o propiedades, de los CD).



**Figura 5-3.** Gramola: un buen ejemplo de objeto independiente

Cuando creas objetos, es mejor utilizar la *encapsulación* o escribir una clase de tal forma que solo se puedan utilizar sus métodos para manipular sus propiedades. En otras palabras, niegas el acceso directo de código externo a sus datos. Los métodos que proporcionas se conocen como la *interfaz* del objeto.

Este enfoque hace que la depuración sea más fácil: tienes que arreglar el código defectuoso solo de una clase. Además, cuando desees actualizar un programa, si has utilizado la encapsulación adecuada y mantienes la misma interfaz, puedes sencillamente desarrollar unas nuevas clases que sustituyan a las antiguas. Si las nuevas no funcionan, puedes volver a cambiarlas por las antiguas para solucionar el problema inmediatamente antes de continuar depurando las nuevas clases.

Una vez que has creado una clase, puedes encontrarte con que necesitas otra clase que sea similar pero no exactamente igual a la primera. Lo más rápido y fácil de hacer es definir una nueva clase usando la *herencia*. Al hacer esto, la nueva clase tiene todas las propiedades de la clase de la que ha recibido la herencia. La clase original ahora se llama *superclase*, y la nueva es la *subclase* (o clase *derivada*).

En nuestro ejemplo de la gramola, si inventas una nueva gramola que puede reproducir un vídeo junto con la música, puede heredar todas las propiedades y métodos de la superclase gramola original y añadir algunas propiedades nuevas (vídeos) y nuevos métodos (un reproductor de películas).

Una excelente ventaja de este sistema es que si se mejora la velocidad o cualquier otro aspecto de la superclase, sus subclases recibirán el mismo beneficio.

## Declaración de clases

Antes de poder utilizar un objeto, debes definir una clase con la palabra clave `class`. La definición de una clase contiene el nombre de la clase (que distingue entre mayúsculas y minúsculas), sus propiedades y sus métodos. En el Ejemplo 5-10 se define la clase `User` con dos propiedades, que son `$name` y `$password` (indicados con la palabra clave `public`, (ver "Ámbito de las propiedades y de los métodos" en la página 111). También crea una nueva instancia (llamada `$object`) de esta clase.

### Ejemplo 5-10. Declaración de una clase y examen de un objeto

```
<?php
    $object = new User; print_r($object);

    class User
    {
        public $name, $password;

        function save_user()
        {
            echo "Save User code goes here";
        }
    }
?>
```

Aquí también he utilizado una función muy importante llamada `print_r`. Le pide a PHP que muestre información sobre una variable de forma que la puedan interpretar las personas. (La `_r` significa *human-readable* [legible por las personas]). En el caso del nuevo objeto `$object`, muestra lo siguiente:

```
User Object (
    [name] =>
    [password] =>
)
```

Sin embargo, un navegador comprime todos los espacios en blanco, por lo que la salida en un navegador es un poco más difícil de leer:

```
User Object ( [name] => [password] => )
```

En cualquier caso, la salida dice que `$object` es un objeto definido por el usuario que tiene las propiedades `name` y `password`.

### Creación de objetos

Para crear un objeto con una clase dada, se usa la palabra clave `new`, así: `$object = new Class`. Aquí hay un par de formas por las que podríamos hacer esto:

```
$object = new User;
$temp   = new User('name', 'password');
```

En la primera línea, sencillamente asignamos un objeto a la clase `User`. En la segunda, pasamos parámetros a la llamada.

Una clase puede requerir o prohibir argumentos; también puede permitir argumentos sin requerirlos explícitamente.

### Acceso a objetos

Añadamos algunas líneas al Ejemplo 5-10 y comprobemos los resultados. En el Ejemplo 5-11 se amplía el código anterior configurando las propiedades del objeto y llamando a un método.

#### Ejemplo 5-11. Creación e interacción con un objeto

```
<?php
    $object = new User;
    print_r($object); echo "<br>";

    $object->name      = "Joe";
    $object->password  = "mypass";
    print_r($object); echo "<br>";

    $object->save_user();

class User
```

## Aprender PHP, MySQL y JavaScript

```
{
    public $name, $password;

    function save_user()
    {
        echo "Save User code goes here";
    }
}
?>
```

Como puedes ver, la sintaxis para acceder a la propiedad de un objeto es `$object->property`. Del mismo modo, se llama a un método de esta forma: `$object->method()`.

Debes tener en cuenta que en el ejemplo, `property` y `method` no tienen el signo `$` delante de ellos. Si antepones el signo `$`, el código no funcionaría, ya que intentaría referenciar el valor dentro de una variable. Por ejemplo, la expresión `$object->$property` intentaría buscar el valor asignado a una variable llamada `$property` (digamos que el valor es la cadena `brown`) y luego intentaría referenciar la propiedad `$object->brown`. Si `$property` no está definida, ocurriría un intento de referencia `$object->NULL` y causaría error.

Cuando se observa utilizando la facilidad View Source de un navegador, la salida del Ejemplo 5-11 es la siguiente:

```
User Object
(
    [name]      =>
    [password] =>
)
User Object
(
    [name]      => Joe
    [password] => mypass
)
Save User code goes here
```

Una vez más, `print_r` muestra su utilidad proporcionando los contenidos de `$object` antes y después de la asignación de la propiedad. De ahora en adelante, omitiré las sentencias `print_r`, pero si estás trabajando con este libro en tu servidor de desarrollo, puedes utilizarla para ver exactamente lo que ocurre.

También puedes ver que el código del método `save_user` se ha ejecutado mediante la llamada a ese método. Ha impreso la cadena recordándonos que debemos crear algún código.



Puedes colocar funciones y definiciones de clase en cualquier lugar de tu código, antes o después de las declaraciones que las usan. Sin embargo, en general se considera una buena práctica colocarlas al final del archivo.

## Clonación de objetos

Una vez creado un objeto, se pasa por referencia cuando lo pasas como un parámetro. En la metáfora de la caja de cerillas, esto es como mantener varios hilos unidos a un objeto almacenado en una caja de fósforos, de modo que puedes seguir cualquier hilo unido a la caja para tener acceso a él.

En otras palabras, hacer la asignación de objetos no copia los objetos en su totalidad.

Puedes ver cómo funciona esto en el Ejemplo 5-12, en el que definimos una clase `User` muy simple, sin métodos y solo con la propiedad `name`.

### Ejemplo 5-12. ¿Copia de un objeto?

```
<?php
$object1      = new User();
$object1->name = "Alice";
$object2      = $object1;
$object2->name = "Amy";

echo "object1 name = " . $object1->name . "<br>";
echo "object2 name = " . $object2->name;

class User
{
    public $name;
}
?>
```

Aquí, primero creamos el objeto `$object1` y le asignamos el valor `Alice` a la propiedad `name`. Luego creamos `$object2`, le asignamos el valor de `$object1`, y asignamos el valor `Amy` solo a la propiedad `name` de `$object2` (o eso es lo que podríamos pensar). Pero este código produce lo siguiente:

```
object1 name = Amy
object2 name = Amy
```

¿Qué ha ocurrido? Tanto `$object1` como `$object2` se refieren al *mismo* objeto, por lo que cambiar la propiedad `name` de `$object2` a `Amy` también establece esa propiedad para `$object1`.

Para evitar esta confusión, puedes utilizar el operador `clone`, que crea una nueva instancia de clase y copia los valores de propiedad de la instancia original a la nueva instancia. El Ejemplo 5-13 ilustra su uso.

### Ejemplo 5-13. Clonación de un objeto

```
<?php
$object1      = new User();
$object1->name = "Alice";
$object2      = clone $object1;
$object2->name = "Amy";
```

## Aprender PHP, MySQL y JavaScript

```
echo "object1 name = " . $object1->name . "<br>";
echo "object2 name = " . $object2->name;

class User
{
    public $name;
}
?>
```

*Voilà!* El resultado de este código es lo que inicialmente queríamos:

```
object1 name = Alice
object2 name = Amy
```

## Constructores

Al crear un nuevo objeto, puedes pasar una lista de argumentos a la clase a la que se llama. Estos se pasan a un método especial dentro de la clase, llamado *constructor*, que inicializa varias propiedades.

Para ello, utiliza la función con nombre `__construct` (es decir, `construct` precedido por dos guiones bajos), como en el Ejemplo 5-14.

**Ejemplo 5-14.** Creación de un método constructor

```
<?php
class User
{
    function __construct($param1, $param2)
    {
        // Constructor statements go here
        public $username = "Guest";
    }
}
?>
```

## Destruyores

También tienes la posibilidad de crear métodos *destruyores*. Esta opción tiene utilidad cuando el código ha hecho la última referencia a un objeto o cuando un script llega al final.

El Ejemplo 5-15 muestra cómo crear un método destructor. El destructor puede hacer labores de limpieza, como liberar una conexión a una base de datos o algún otro recurso que has reservado dentro de la clase. Puesto que has reservado el recurso dentro de la clase, tienes que liberarlo aquí o se quedará en ella indefinidamente. Muchos problemas en el conjunto del sistema los causan programas que ocupan recursos y se olvida liberarlos.

**Ejemplo 5-15.** Creación de un método destructor

```
<?php
class User
```



```
{
    function __destruct()
    {
        // Destructor code goes here
    }
}
?>
```

### Métodos de escritura

Como hemos visto, declarar un método es similar a declarar una función, pero hay algunas diferencias. Por ejemplo, los nombres de métodos que empiezan con un doble guion bajo (\_\_) están reservados, y no deberías crear ninguno que respondiera a este formato.

También tienes acceso a una variable especial llamada `$this`, que puede utilizarse para acceder a las propiedades del objeto actual. Para ver cómo funciona, echa un vistazo al Ejemplo 5-16, que contiene un método diferente de la definición de clase `User` llamado `get_password`.

#### Ejemplo 5-16. Uso de la variable `$this` en un método

```
<?php
class User
{
    public $name, $password;

    function get_password()
    {
        return $this->password;
    }
}
?>
```

`get_password` usa la variable `$this` para acceder al objeto actual y luego devuelve el valor de la propiedad `password` de ese objeto. Observa cómo se omite el signo `$` precedente de la propiedad `$password` cuando usamos el operador `->`. No eliminar el signo `$` es un error típico con el que puedes encontrarte, particularmente cuando usas esta característica por primera vez.

He aquí cómo se usaría la clase definida en el Ejemplo 5-16:

```
$object          = new User;
$object->password = "secret";

echo $object->get_password();
```

Este código imprime la contraseña `secret`.

### Declaración de propiedades

No es necesario declarar explícitamente las propiedades dentro de las clases, ya que pueden definirse implícitamente cuando se utilizan por primera vez. Para ilustrar esto, en el Ejemplo 5-17 la clase `User` no tiene propiedades ni métodos, pero es un código permitido.

#### Ejemplo 5-17. Definición implícita de una propiedad

```
<?php
    $object1          = new User();
    $object1->name = "Alice";

    echo $object1->name;

    class User {}
?>
```

Este código produce correctamente la cadena `Alice` sin ningún problema, porque PHP implícitamente declara la propiedad `$object1->name` en tu lugar. Pero este tipo de programación puede conducir a errores que son exasperantemente difíciles de descubrir, porque `name` se ha declarado fuera de la clase.

Para ayudarte y ayudar a cualquier otra persona que mantenga tu código, te aconsejo que adquieras el hábito de declarar siempre tus propiedades explícitamente dentro de las clases. Te alegrarás de haberlo hecho.

Además, cuando se declara una propiedad dentro de una clase, se le puede asignar un valor por defecto. El valor que utilices debe ser una constante y no el resultado de una función o expresión. El Ejemplo 5-18 muestra algunas asignaciones válidas y otras que no lo son.

#### Ejemplo 5-18. Declaraciones de propiedades válidas y no válidas.

```
<?php
class Test
{
    public $name = "Paul Smith"; // Valid
    public $age  = 42;           // Valid
    public $time = time();       // Invalid - calls a function
    public $score = $level * 2;  // Invalid - uses an expression
}
?>
```

### Declaración de constantes

De la misma manera que se puede crear una constante global con la función `define`, se pueden definir constantes dentro de clases. La práctica generalmente aceptada es usar letras mayúsculas para que destaquen, como en el Ejemplo 5-19.

### Ejemplo 5-19. Definición de constantes dentro de una clase

```
<?php
    Translate::lookup();

    class Translate
    {
        const ENGLISH = 0;
        const SPANISH = 1;
        const FRENCH = 2;
        const GERMAN = 3;
        // ...

        static function lookup()
        {
            echo self::SPANISH;
        }
    }
?>
```

Puedes referenciar las constantes directamente, mediante la palabra clave `self` y el operador de dos puntos. Ten en cuenta que este código llama a la clase directamente, utilizando el operador de dos puntos en la línea 1, sin crear antes una instancia del mismo. Como era de esperar, el resultado cuando ejecutes este código es 1.

Recuerda que una vez que defines una constante, no puedes cambiarla.

## Ámbito de las propiedades y de los métodos

PHP proporciona tres palabras clave para controlar el campo de acción de las propiedades y métodos (*miembros*):

### Public (pública)

Los miembros con clave pública se pueden referenciar en cualquier lugar, incluso por otras clases e instancias del objeto. Este es el valor por defecto cuando las variables se declaran con las palabras clave `var` o `public`, o cuando una variable se declara implícitamente la primera vez que se utiliza. Las palabras clave `var` y `public` son intercambiables porque, aunque obsoleta, `var` mantiene la compatibilidad con versiones anteriores de PHP. Se supone que los métodos son `public` por defecto.

### Protected (protegida)

Estos miembros se pueden referenciar solo por los métodos de clase del objeto y los de cualquier subclase.

### Private (privada)

Estos miembros se pueden ser referenciar solo por métodos dentro de la misma clase, no por subclases.

## Aprender PHP, MySQL y JavaScript

He aquí cómo decidir qué necesitas usar:

- Usa `public` cuando el código externo *debe* acceder a este miembro y la extensión de clases también *debe* heredarlo.
- Usa `protected` cuando el código externo *no debe* acceder a este miembro, pero la extensión de clases *debe* heredarlo.
- Usa `private` cuando el código externo *no debe* acceder a este miembro y la extensión de clases *tampoco debe* heredarlo.

El Ejemplo 5-20 ilustra el uso de estas palabras clave.

### Ejemplo 5-20. Cambio del campo de acción de las propiedades y los métodos

```
<?php
class Example
{
    var $name    = "Michael";    // Same as public but deprecated
    public $age  = 23;           // Public property
    protected $usercount;       // Protected property

    private function admin()    // Private method
    {
        // Admin code goes here
    }
}
?>
```

## Métodos estáticos

Podemos definir un método como `static` (estático), lo que significa que puede ser invocado por una clase, no por un objeto. Un método estático no tiene acceso a ninguna propiedad de objeto y se crea y se accede a él como se ve en el Ejemplo 5-21.

### Ejemplo 5-21. Creación y acceso a un método estático

```
<?php
User::pwd_string();

class User
{
    static function pwd_string()
    {
        echo "Please enter your password";
    }
}
?>
```

Observa cómo llamamos a la clase misma, junto con el método estático, mediante los dos puntos (también conocido como operador de *resolución de ámbito*), en lugar de `->`. Las funciones estáticas son útiles para realizar acciones relacionadas con la clase misma, pero no con instancias específicas de la clase.

Puedes ver otra muestra de un método estático en el Ejemplo 5-19.



Si tratas de acceder a `$this->property` o a otras propiedades del objeto desde una función estática, se presentará un mensaje de error.

### Propiedades estáticas

La mayoría de los datos y métodos se aplican a las instancias de una clase. Por ejemplo, en una clase `User`, deseas hacer cosas tales como establecer la contraseña de un usuario en particular o comprobar cuándo se ha registrado. Estas acciones y operaciones se aplican por separado a cada usuario y por lo tanto utilizan propiedades y métodos específicos de cada instancia.

Pero ocasionalmente querrás mantener los datos de toda una clase. Por ejemplo, para informar de cuántos usuarios están registrados, almacenarás una variable que se aplica a toda la clase `User`. PHP proporciona propiedades estáticas y métodos para tales datos.

Como se muestra brevemente en el Ejemplo 5-21, la declaración de miembros de una clase `static` los hace accesibles sin una instanciación de la clase. A una propiedad declarada `static` no puede acceder directamente una instancia de una clase, pero un método estático puede hacerlo.

El Ejemplo 5-22 define una clase llamada `Test` con una propiedad estática y un método público.

#### Ejemplo 5-22. Definición de una clase con propiedad estática

```
<?php
$temp = new Test();

echo "Test A: " . Test::$static_property . "<br>";
echo "Test B: " . $temp->get_sp() . "<br>";
echo "Test C: " . $temp->static_property . "<br>";

class Test
{
    static $static_property = "I'm static";

    function get_sp()
    {
        return self::$static_property;
    }
}
?>
```

Cuando se ejecuta este código, devuelve la siguiente salida:

```
Test A: I'm static
Test B: I'm static
```

## Aprender PHP, MySQL y JavaScript

**Notice: Undefined property: Test::\$static\_property  
Test C:**

Este ejemplo muestra que la propiedad `$static_property` podría ser referenciada directamente desde la clase misma a través del operador de dos puntos en Test A. Además, Test B podría obtener su valor llamando al método `get_sp` del objeto `$temp`, creado a partir de la clase Test. Pero Test C falló porque la propiedad estática `$static_property` no era accesible para el objeto `$temp`.

Observa cómo el método `get_sp` accede a `$static_property` mediante la palabra clave `self`. Así es como se puede acceder directamente a una propiedad estática o a una constante dentro de una clase.

## Herencia

Una vez que hayas escrito una clase, puedes derivar subclases de ella. Esto puede ahorrar mucho trabajo de reescritura de código: puedes tomar una clase similar a la que necesitas escribir, extenderla a una subclase y solo modificar las partes que son diferentes. Esto se consigue mediante la palabra clave `extends`.

En el Ejemplo 5-23, la clase `Subscriber` se declara una subclase de `User` mediante la palabra clave `extends`.

### Ejemplo 5-23. Herencia y extensión de una clase

```
<?php
$object                = new Subscriber;
$object->name           = "Fred";
$object->password       = "pword";
$object->phone          = "012 345 6789";
$object->email          = "fred@bloggs.com";
$object->display();

class User
{
    public $name, $password;

    function save_user()
    {
        echo "Save User code goes here";
    }
}

class Subscriber extends User
{
    public $phone, $email;

    function display()
    {
        echo "Name:  " . $this->name      . "<br>";
        echo "Pass:  " . $this->password . "<br>";
    }
}
```

## 5. Funciones y objetos en PHP

```
        echo "Phone: " . $this->phone . "<br>";
        echo "Email: " . $this->email;
    }
}
?>
```

La clase original `User` tiene dos propiedades, `$name` y `$password`, y un método para guardar al usuario actual en la base de datos. `Subscriber` amplía esta clase añadiendo dos propiedades adicionales, `$phone` y `$email`, e incluye un método para mostrar las propiedades del objeto actual mediante la variable `$this`, que se refiere a los valores actuales del objeto al que se accede. La salida de este código es la siguiente:

```
Name: Fred
Pass: pword
Phone: 012 345 6789
Email: fred@bloggs.com
```

### La palabra clave `parent`

Si escribes un método en una subclase con el mismo nombre que una de su clase padre, sus declaraciones prevalecerán sobre las de la clase padre. A veces este no es el comportamiento que deseas, y necesitas tener acceso al método padre. Para ello, puedes utilizar el operador `parent`, como en el Ejemplo 5-24.

#### Ejemplo 5-24. Redefinición de un método y uso del operador `parent`

```
<?php
$object = new Son;
$object->test();
$object->test2();

class Dad
{
    function test()
    {
        echo "[Class Dad] I am your Father<br>";
    }
}

class Son extends Dad
{
    function test()
    {
        echo "[Class Son] I am Luke<br>";
    }

    function test2()
    {
        parent::test();
    }
}
?>
```

## Aprender PHP, MySQL y JavaScript

Este código crea una clase llamada `Dad` y una subclase llamada `Son`, que hereda sus propiedades y métodos, y luego anula el método `test`. Por lo tanto, cuando la línea 2 llama al método `test`, se ejecuta el nuevo método. La única manera de ejecutar el método `test` redefinido en la clase `Dad` es utilizar el operador `parent`, como se muestra en `test2` de la clase `Son`. El código proporciona la salida siguiente:

```
[Class Son] I am Luke
[Class Dad] I am your Father
```

Si quieres tener la seguridad de que tu código llama a un método de la clase actual, puedes utilizar la palabra clave `self`, así:

```
self::method();
```

### Constructores de subclase

Cuando extiendas una clase y declares tu propio constructor, debes tener en cuenta que PHP no llamará automáticamente al método constructor de la clase padre. Si quieres estar seguro de que se ejecuta todo el código de inicialización, las subclases siempre deben llamar a los constructores padre, como en el Ejemplo 5-25.

#### Ejemplo 5-25. Llamada al constructor de la clase `parent`

```
<?php
$object = new Tiger();

echo "Tigers have...<br>";
echo "Fur: " . $object->fur . "<br>";
echo "Stripes: " . $object->stripes;

class Wildcat
{
    public $fur; // Wildcats have fur

    function __construct()
    {
        $this->fur = "TRUE";
    }
}

class Tiger extends Wildcat
{
    public $stripes; // Tigers have stripes

    function __construct()
    {
        parent::__construct(); // Call parent constructor first
        $this->stripes = "TRUE";
    }
}

?>
```



Este ejemplo aprovecha la herencia de la forma habitual. La clase `Wildcat` ha creado la propiedad `$fur`, que nos gustaría reutilizar, así que creamos la clase `Tiger` para heredar `$fur` y además crear otra propiedad, `$stripes`. Para verificar que se ha llamado a ambos constructores, el programa da salida a lo siguiente:

```
Tigers have...  
Fur: TRUE  
Stripes: TRUE
```

### Método final

Cuando quieras evitar que una subclase anule un método de superclase, puedes utilizar la palabra clave `final`. El Ejemplo 5-26 muestra como.

#### Ejemplo 5-26. Creación de un método final

```
<?php  
class User  
{  
    final function copyright()  
    {  
        echo "This class was written by Joe Smith";  
    }  
}  
?>
```

Una vez hayas digerido el contenido de este capítulo, debes tener una idea clara de lo que PHP puede hacer por ti. Deberías poder utilizar las funciones con facilidad y, si lo deseas, escribir código orientado a objetos. En el Capítulo 6, terminaremos nuestra exploración inicial de PHP examinando el funcionamiento de las matrices PHP.

## Preguntas

1. ¿Cuál es la principal ventaja de utilizar una función?
2. ¿Cuántos valores puede devolver una función?
3. ¿Cuál es la diferencia entre acceder a una variable por nombre y por referencia?
4. ¿Cuál es el significado de *scope* en PHP?
5. ¿Cómo se puede incorporar un archivo PHP dentro de otro?
6. ¿En qué se diferencia un objeto de una función?
7. ¿Cómo se crea un nuevo objeto en PHP?
8. ¿Qué sintaxis usarías para crear una subclase a partir de una existente?
9. ¿Cómo se puede hacer que un objeto se inicialice cuando lo creas?
10. ¿Por qué es una buena idea declarar explícitamente las propiedades dentro de una clase?

## **Aprender PHP, MySQL y JavaScript**

Consulta “"Respuestas del Capítulo 5" en la página 708 en el Apéndice A para comprobar las respuestas a estas preguntas.

## CAPÍTULO 6

# Matrices en PHP

En el Capítulo 3, hice una introducción muy breve a las matrices en PHP, lo suficiente para saborear ligeramente sus posibilidades. En este capítulo, te mostraré muchas más cosas que puedes hacer con matrices, algunas de las cuales, si alguna vez has utilizado un lenguaje fuertemente tipado, como puede ser C, pueden sorprenderte por su elegancia y sencillez.

Las matrices son un ejemplo de lo que ha hecho tan popular a PHP. No solo eliminan el tedio de escribir código cuando tratamos con estructuras de datos complicadas, sino que también proporcionan numerosas formas de acceder a los datos sin dejar de ser un proceso increíblemente rápido.

## Introducción

Ya nos hemos referido a las matrices como si fueran grupos de cajas de cerillas pegadas entre sí. Otra manera de imaginar una matriz es como si se tratara de una cadena de cuentas, en la que las cuentas pueden representar números, cadenas de caracteres o incluso otras matrices. Son como cuerdas de cuentas en las que cada elemento tiene su propia ubicación y (con la excepción del primero y el último) cada uno tiene otros elementos a cada lado.

Algunas matrices están referenciadas por índices numéricos; otras permiten identificadores alfanuméricos. Las funciones integradas permiten clasificarlas, agregar o quitar secciones y recorrerlas para manejar cada elemento mediante un tipo especial de bucle. Y si colocas una o más dentro de otra, puedes crear matrices de dos, tres o de cualquier número de dimensiones.

## Matrices indexadas numéricamente

Supongamos que se te ha encomendado la tarea de crear un sitio web sencillo para una oficina local de suministros de oficina, y que te encuentras desarrollando la sección dedicada al papel. Una de las formas de gestionar los distintos artículos de esta categoría sería colocarlos en una matriz numérica. Puedes ver la forma más sencilla de hacerlo en el Ejemplo 6-1.

## Aprender PHP, MySQL y JavaScript

### Ejemplo 6-1. Adición de elementos a una matriz

```
<?php
    $paper[] = "Copier";
    $paper[] = "Inkjet";
    $paper[] = "Laser";
    $paper[] = "Photo";

    print_r($paper);
?>
```

En este ejemplo, cada vez que asignas un valor a la matriz `$paper`, se almacena en la primera ubicación vacía dentro de la matriz, y se incrementa un puntero interno de PHP para apuntar a la siguiente ubicación libre, que queda preparada para futuras inserciones. La conocida función `print_r` (que imprime el contenido de una variable, matriz u objeto) se utiliza para verificar que la matriz se ha llenado correctamente. La función imprime lo siguiente:

```
Array
(
    [0] => Copier
    [1] => Inkjet
    [2] => Laser
    [3] => Photo
)
```

El código anterior también podría haberse escrito como se muestra en el Ejemplo 6-2, en el que se especifica la ubicación exacta de cada elemento dentro de la matriz. Pero, como puedes ver, ese enfoque requiere escritura extra y hace que el código sea más difícil de mantener si quieres añadir o eliminar suministros de la matriz. Por lo tanto, a menos que quieras especificar un orden diferente, por lo general, es mejor dejar que PHP gestione los números de ubicación reales.

### Ejemplo 6-2. Adición de elementos a una matriz utilizando ubicaciones explícitas

```
<?php
    $paper[0] = "Copier";
    $paper[1] = "Inkjet";
    $paper[2] = "Laser";
    $paper[3] = "Photo";

    print_r($paper);
?>
```

El resultado de estos ejemplos es idéntico, pero no es probable que utilices `print_r` en un sitio web en producción, por lo que el Ejemplo 6-3 muestra cómo podrías imprimir los distintos tipos de papel que ofrece el sitio web con un bucle `for`.

### Ejemplo 6-3. Adición y recuperación de elementos de una matriz

```
<?php
    $paper[] = "Copier";
    $paper[] = "Inkjet";
    $paper[] = "Laser";
    $paper[] = "Photo";

    for ($j = 0 ; $j < 4 ; ++$j)
        echo "$j: $paper[$j]<br>";
?>
```

Este ejemplo imprime lo siguiente:

```
0: Copier
1: Inkjet
2: Laser
3: Photo
```

Hasta ahora, has visto un par de procedimientos con los que puedes añadir elementos a una matriz y una manera de referenciarlos. PHP ofrece muchas más cosas, a las que llegaré en breve. Pero antes veamos otro tipo de matriz.

## Matrices asociativas

Hacer un seguimiento de los elementos de la matriz por índices funciona bien, pero puede requerir trabajo extra en términos de recordar qué número se refiere a qué producto. También puede hacer que otros programadores tengan dificultades para seguir el código.

Aquí es donde las matrices asociativas entran en juego. Con ellas puedes hacer referencia a los elementos de la matriz por nombres en lugar de hacerlo por números. El Ejemplo 6-4 amplía el código anterior dando a cada elemento de la matriz un nombre identificativo y un valor de la cadena de caracteres más largo y más explicativo.

### Ejemplo 6-4. Adición y recuperación de elementos de una matriz asociativa

```
<?php
    $paper['copier'] = "Copier & Multipurpose";
    $paper['inkjet'] = "Inkjet Printer";
    $paper['laser']  = "Laser Printer";
    $paper['photo']  = "Photographic Paper";

    echo $paper['laser'];
?>
```

En lugar de un número (que no ofrece ninguna información útil, excepto la posición del elemento en la matriz), cada elemento tiene ahora un nombre único que puedes utilizar para referenciarlo en otro lugar, como en la declaración `echo`, que imprime solamente `LaserPrinter`. Los nombres (`copier`, `inkjet`, etc.) se denominan *índices* o *claves*, y los elementos asignados a ellos (como `Laser Printer`) se denominan *valores*.

## Aprender PHP, MySQL y JavaScript

Esta importante característica de PHP se utiliza a menudo cuando se extrae información de XML y HTML. Por ejemplo, un analizador HTML como los que utilizan los motores de búsqueda podría colocar todos los elementos de una página web en una matriz asociativa cuyos nombres reflejen la estructura de la página:

```
$html['title'] = "My web page";  
$html['body'] = "... body of web page ...";
```

El programa probablemente también desglosaría todos los enlaces encontrados dentro de una página en otra matriz, y todos los encabezados y subencabezados, en otra. Cuando utilizas matrices asociativas en lugar de matrices numéricas, el código para referirse a todos estos elementos es fácil de escribir y de depurar.

## Asignación mediante la palabra clave array

Hasta ahora, hemos visto cómo asignar valores a las matrices añadiendo nuevos elementos de uno en uno. Tanto si especificas claves, como si especificas identificadores numéricos o dejas que PHP asigne identificadores numéricos implícitamente, es una solución que requiere mucho tiempo. Un sistema más compacto y un método de asignación más rápido emplea la palabra clave array. El Ejemplo 6-5 muestra la asignación de una matriz tanto numérica como asociativa con este método.

**Ejemplo 6-5.** Adición de elementos a una matriz utilizando la palabra clave array

```
<?php  
$p1 = array("Copier", "Inkjet", "Laser", "Photo");  
  
echo "p1 element: " . $p1[2] . "<br>";  
  
$p2 = array('copier' => "Copier & Multipurpose",  
            'inkjet'  => "Inkjet Printer",  
            'laser'   => "Laser Printer",  
            'photo'   => "Photographic Paper");  
  
echo "p2 element: " . $p2['inkjet'] . "<br>";  
?>
```

La primera parte de este fragmento de código asigna las antiguas y breves descripciones del producto a la matriz `$p1`. Hay cuatro elementos, por lo que ocuparán las ranuras de 0 a 3. Por lo tanto, la sentencia `echo` imprime lo siguiente:

```
p1 element: Laser
```

La segunda parte asigna identificadores asociativos y descripciones de producto más largas a la matriz `$p2`, mediante el formato `key => value`. El uso de `=>` es similar al operador de asignación normal `=`, excepto que se está asignando un valor a un *índice* y no a una *variable*. El índice está entonces inextricablemente ligado a ese valor, a menos que se asigne un nuevo valor. Por lo tanto, el comando `echo` imprime esto:

```
p2 element: Inkjet Printer
```

Puedes comprobar que `$p1` y `$p2` son diferentes tipos de matrices, porque los dos comandos siguientes, cuando se añaden al código, darán lugar a un `Undefined index` (índice indefinido) o un error de `Undefined offset` (desplazamiento indefinido), ya que el identificador de la matriz para cada uno de ellos es incorrecto:

```
echo $p1['inkjet'];    // Undefined index
echo $p2[3];          // Undefined offset
```

## Bucle foreach...as

Los creadores de PHP han hecho todo lo posible para que el lenguaje sea fácil de usar. Así que, no contentos con las estructuras de bucle ya incorporadas, añadieron otra especial para matrices: el bucle `foreach...as`. Si lo utilizas, puedes desplazarte por todos los elementos de una matriz, uno a continuación de otro, y hacer lo que necesites con ellos.

El proceso comienza con el primer elemento y termina con el último, por lo que no tienes que saber cuántos elementos hay en una matriz. El Ejemplo 6-6 muestra cómo podemos utilizar `foreach...as` para reescribir el Ejemplo 6-3.

**Ejemplo 6-6.** Recorrido a través de una matriz numérica usando `foreach...as`

```
<?php
    $paper = array("Copier", "Inkjet", "Laser", "Photo");
    $j = 0;

    foreach($paper as $item)
    {
        echo "$j: $item<br>";
        ++$j;
    }
?>
```

Cuando PHP encuentra una instrucción `foreach`, toma el primer elemento de la matriz y lo coloca en la variable que sigue a la palabra clave `as`; y cada vez que el flujo de control vuelve a `foreach`, el siguiente elemento de la matriz se coloca en la variable que sigue a la palabra clave `as`. En este caso, el valor de la variable `$item` se fija sucesivamente con cada uno de los cuatro valores de la matriz `$paper`. Una vez que se han utilizado todos los valores, termina la ejecución del bucle. La salida de este código es exactamente igual que la del Ejemplo 6-3.

Ahora veamos cómo `foreach` trabaja con una matriz asociativa en el Ejemplo 6-7, que es una reescritura de la segunda parte del Ejemplo 6-5.

**Ejemplo 6-7.** Recorrido a través de una matriz asociativa utilizando `foreach...as`

```
<?php
    $paper = array('copier' => "Copier & Multipurpose",
                  'inkjet' => "Inkjet Printer",
```

## Aprender PHP, MySQL y JavaScript

```
        'laser' => "Laser Printer",
        'photo' => "Photographic Paper");

foreach($paper as $item => $description)
    echo "$item: $description<br>";
?>
```

Recuerda que las matrices asociativas no requieren índices numéricos, por lo que la variable `$j` no se utiliza en este ejemplo. En su lugar, cada elemento de la matriz `$paper` se introduce en el par clave/valor de las variables `$item` y `$description`, desde donde se imprimen.

El resultado visualizado de este código es el siguiente:

```
copier: Copier & Multipurpose
inkjet: Inkjet Printer
laser: Laser Printer
photo: Photographic Paper
```

Como `list` es una sintaxis alternativa a `foreach...as`, puedes utilizarla junto con la función `each`, como en el Ejemplo 6-8.

### Ejemplo 6-8. Recorrido a través de una matriz asociativa utilizando `each` y `list`

```
<?php
    $paper = array('copier' => "Copier & Multipurpose",
                  'inkjet' => "Inkjet Printer",
                  'laser' => "Laser Printer",
                  'photo' => "Photographic Paper");

    while (list($item, $description) = each($paper))
        echo "$item: $description<br>";
?>
```

En este ejemplo, se configura un bucle `while` y continuará ejecutándose hasta que `each` devuelva el valor `FALSE`. La función `each` actúa como `foreach`: devuelve una matriz que contiene el par clave/valor de la matriz `$paper` y luego mueve el puntero integrado al siguiente par de esa matriz. Cuando no hay más pares que devolver, `each` devuelve `FALSE`.

La función `list` utiliza una matriz como argumento (en este caso, el par clave/valor devuelto por la función `each`) y luego asigna los valores de la matriz a las variables listadas entre paréntesis.

Puedes ver cómo funciona `list` con un poco más de detalle en el Ejemplo 6-9, en el que se crea una matriz a partir de las dos cadenas `Alice` y `Bob`, y luego se pasa a la función `list`, que asigna esas cadenas como valores a las variables `$a` y `$b`.

### Ejemplo 6-9. Uso de la función `list`

```
<?php
    list($a, $b) = array('Alice', 'Bob');
    echo "a=$a b=$b";
?>
```



La salida de este código es la siguiente:

```
a=Alice b=Bob
```

Por lo tanto, puedes elegir el recorrido por las matrices. Usa `foreach...as` para crear un bucle que extrae valores y los coloca en la variable que sigue a `as`, o usa la función `each` y crea tu propio sistema de bucle.

## Matrices de varias dimensiones

Una sencilla característica de diseño en la sintaxis de matrices en PHP hace posible crear matrices de más de una dimensión. De hecho, pueden tener tantas dimensiones como quieras (aunque es raro encontrar una aplicación que utilice más de tres).

Esta característica es la capacidad de incluir una matriz completa como parte de otra, y es posible repetir el proceso, como en la vieja rima: "Las pulgas grandes tienen pulgas más pequeñas sobre sus espaldas, que las muerden. Las pulgas pequeñas tienen pulgas más pequeñas, y estas otras más pequeñas, *ad infinitum*".

Veamos cómo funciona esta operación en la matriz asociativa del ejemplo anterior y la ampliamos. Ver el Ejemplo 6-10.

### Ejemplo 6-10. Creación de una matriz asociativa de varias dimensiones

```
<?php
$products = array(
    'paper' => array(
        'copier' => "Copier & Multipurpose",
        'inkjet' => "Inkjet Printer",
        'laser'  => "Laser Printer",
        'photo'  => "Photographic Paper"),
    'pens' => array(
        'ball'   => "Ball Point",
        'hilite' => "Highlighters",
        'marker' => "Markers"),
    'misc' => array(
        'tape'   => "Sticky Tape",
        'glue'   => "Adhesives",
        'clips'  => "Paperclips"
    )
);
echo "<pre>";

foreach($products as $section => $items)
    foreach($items as $key => $value)
        echo "$section:\t$key\t($value)<br>";

echo "</pre>";
?>
```

## Aprender PHP, MySQL y JavaScript

Para aclarar las cosas, ahora que el código está empezando a crecer, he cambiado el nombre de algunos de los elementos. Por ejemplo, debido a que la matriz anterior `$paper` es ahora solo una subsección de una matriz más grande, la matriz principal se llama ahora `$products`. Dentro de esta matriz, hay tres elementos (`paper`, `pens` y `misc`) cada uno de los cuales contiene otra matriz con pares clave/valor.

Si fuera necesario, estas submatrices podrían haber contenido aún más matrices. Por ejemplo, en `ball` puede haber muchos tipos y colores diferentes de bolígrafos disponibles en la tienda online. Pero por ahora, he restringido el código a un nivel de solo dos.

Una vez que se han asignado los datos de la matriz, utilizo un par de bucles `foreach...as` anidados para imprimir los diferentes valores. El bucle exterior extrae las secciones principales del nivel superior de la matriz, y el bucle interior extrae los pares clave/valor para las categorías dentro de cada sección.

Si recuerdas que cada nivel de la matriz funciona de la misma manera (es un par clave/valor), puedes escribir fácilmente el código para acceder a cualquier elemento en cualquier nivel.

La declaración `echo` hace uso del carácter escape `\t` de PHP, que crea una tabulación.

Aunque las tabulaciones no son normalmente importantes para el navegador web, las empleo para maquetación, utilizo las etiquetas `<pre>...</pre>`, que le indican al navegador web que debe dar formato al archivo de texto como preformateado y monoespaciado, y que *no* ignore caracteres de espacios en blanco tales como como tabulaciones y saltos de línea. La salida de este código tiene el siguiente aspecto:

```
paper: copier    (Copier & Multipurpose)
paper: inkjet    (Inkjet Printer)
paper: laser     (Laser Printer)
paper: photo     (Photographic Paper)
pens:  ball      (Ball Point)
pens:  hilite    (Highlighters)
pens:  marker    (Markers)
misc:  tape      (Sticky Tape)
misc:  glue      (Adhesives)
misc:  clips     (Paperclips)
```

Puedes acceder directamente a un elemento particular de la matriz usando corchetes:

```
echo $products['misc']['glue'];
```

Esto proporciona como salida el valor `Adhesives`.

También puedes crear matrices numéricas de varias dimensiones a las que acceden directamente los índices en lugar de los identificadores alfanuméricos. En el Ejemplo 6-11 se crea el tablero de un juego de ajedrez con las piezas situadas en sus posiciones iniciales.

**Ejemplo 6-11.** Creación de una matriz numérica de varias dimensiones

```
<?php
$chessboard = array(
    array('r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'),
    array('p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array('P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'),
    array('R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R')
);

echo "<pre>"; foreach($chessboard as $row)
{
    foreach ($row as $piece)
        echo "$piece ";

    echo "<br>";
}

echo "</pre>";
?>
```

En este ejemplo, las letras minúsculas representan las piezas negras y las mayúsculas, las blancas.

La clave es `r` = rook (torre), `n` = knight (caballo), `b` = bishop (alfil), `k` = king (rey), `q` = queen (reina) y `p` = pawn (peón). Otra vez, un par de bucles `foreach...as` anidados recorren la matriz y muestra su contenido. El bucle exterior procesa cada fila en la variable `$row`, que a su vez es una matriz porque la matriz `$chessboard` usa una submatriz para cada fila. Este bucle contiene dos declaraciones, encerradas entre llaves.

El bucle interior procesa entonces cada cuadrado de una fila y da salida al carácter (`$piece`), seguido de un espacio (para cuadrar la impresión). Este bucle tiene una sola declaración, por lo que no se requieren llaves para cerrarla. `<pre>` y `</pre>` aseguran que la salida se muestre correctamente, así:

```

r n b q k b n r
p p p p p p p

P P P P P P P
R N B Q K B N R
```

También puedes acceder directamente a cualquier elemento dentro de esta matriz usando corchetes:

```
echo $chessboard[7][3];
```

Esta declaración da como resultado la letra mayúscula Q, el octavo elemento hacia abajo y el cuarto a lo largo (recuerda que los índices de la matriz comienzan en 0, no en 1).

## Uso de funciones en matrices

Ya has visto las funciones `list` y `each`, pero PHP viene equipado con un buen número de otras funciones con las que tratar matrices. Puedes encontrar la lista completa en la documentación (<http://php.net/manual/es/ref.array.php>). Sin embargo, algunas de estas funciones son tan importantes que vale la pena dedicarle tiempo y estudiarlas aquí.

### `is_array`

Las matrices y variables comparten el mismo espacio de nombres. Esto significa que no es posible tener una variable de cadena de caracteres llamada `$fred` y una matriz también llamada `$fred`. Si tienes alguna duda y tu código necesita comprobar si una variable es una matriz, puedes usar la función `is_array`, así:

```
echo (is_array($fred)) ? "Is an array" : "Is not an array";
```

Ten en cuenta que si todavía no se ha asignado un valor a `$fred`, se generará el mensaje `Undefined variable`.

### `count`

Aunque la función `each` y la estructura del bucle `foreach`...as son excelentes maneras de recorrer el contenido de una matriz, a veces necesitas saber exactamente cuántos elementos hay, particularmente si los vas a referenciar directamente. Para contar todos los elementos en el nivel superior de una matriz, utiliza un comando como este:

```
echo count($fred);
```

Si deseas saber cuántos elementos hay en total en una matriz de varias dimensiones, puedes utilizar una expresión como la siguiente:

```
echo count($fred, 1);
```

El segundo parámetro es opcional y configura el modo a utilizar. Debe ser 0 para limitar el cómputo solo al nivel superior o 1 para forzar el cómputo recursivo de todos los elementos de la submatriz también.

### `sort`

La ordenación es tan habitual que PHP proporciona una función integrada para ella. En su forma más elemental, se usaría así:

```
sort($fred);
```

A diferencia de otras funciones, `sort` actuará directamente sobre la matriz suministrada en lugar de devolver una nueva matriz de elementos ordenados. Devuelve `TRUE` en caso de éxito y `FALSE` en caso de error, y también utiliza algunos indicadores. Los dos principales que puedes usar para forzar que los elementos se ordenen bien numéricamente o como cadenas son los siguientes:

```
sort($fred, SORT_NUMERIC); sort($fred, SORT_STRING);
```

También puedes ordenar una matriz en orden inverso mediante la función `rsort`, así:

```
rsort($fred, SORT_NUMERIC); rsort($fred, SORT_STRING);
```

### shuffle

Puede haber ocasiones en las que necesites que los elementos de una matriz tengan una distribución aleatoria, como cuando juegas a las cartas:

```
shuffle($cards);
```

Como `sort`, `shuffle` actúa directamente sobre la matriz suministrada y devuelve `TRUE` en caso de éxito o `FALSE` en caso de error.

### explode

`explode` es una función muy útil con la que puedes actuar sobre una cadena que contenga varios elementos separados por un solo carácter (o por una cadena de caracteres) y colocar cada uno de estos elementos en una matriz. Un ejemplo práctico es dividir una oración en una matriz que contenga todas sus palabras, como en el Ejemplo 6-12.

**Ejemplo 6-12.** Separación de una cadena en una matriz por medio de espacios

```
<?php
$temp = explode(' ', "This is a sentence with seven words");
print_r($temp);
?>
```

Este ejemplo imprime lo siguiente (en una sola línea cuando se ve en un navegador):

```
Array (
    [0] => This
    [1] => is
    [2] => a
    [3] => sentence
    [4] => with
    [5] => seven
    [6] => words
)
```

## Aprender PHP, MySQL y JavaScript

El primer parámetro, el delimitador, no necesita ser un espacio ni incluso un solo carácter. El Ejemplo 6-13 muestra una ligera variación.

**Ejemplo 6-13.** Separación de una cadena delimitada con **\*\*\*** en una matriz

```
<?php
    $temp = explode('***', "A***sentence***with***asterisks");
    print_r($temp);
?>
```

El código del Ejemplo 6-13 imprime lo siguiente:

```
Array (
    [0] => A
    [1] => sentence
    [2] => with
    [3] => asterisks
)
```

### extract

A veces puede ser conveniente convertir los pares clave/valor de una matriz en variables PHP. Una de esas veces podría ser cuando procesas las variables `$_GET` o `$_POST` enviadas a un script PHP para un formulario.

Cuando se envía un formulario a través de la web, el servidor web descomprime las variables en una matriz global para el script PHP. Si las variables se han enviado con el método GET, se colocarán en una matriz asociativa llamada `$_GET`; si se han enviado con POST, se colocarán en una matriz asociativa llamada `$_POST`.

Podrías, por supuesto, recorrer las matrices asociativas de la manera que se muestra en los ejemplos presentados hasta ahora. Sin embargo, a veces solo quieres almacenar los valores enviados para su uso posterior. En este caso, puede hacer que PHP haga este trabajo automáticamente:

```
extract($_GET);
```

Por lo tanto, si el parámetro `q` de la cadena de consulta se envía a un script PHP junto con el valor asociado `Hi there`, se creará una nueva variable llamada `$q` y se le asignará ese valor.

Sin embargo, ten cuidado con este enfoque, porque si alguna de las variables extraídas entra en conflicto con las que ya has definido, los valores existentes se sobrescribirán. Para evitar esta posibilidad, puedes utilizar uno de los muchos parámetros adicionales disponibles para esta función, así:

```
extract($_GET, EXTR_PREFIX_ALL, 'fromget');
```

En este caso, todas las nuevas variables comenzarán con la cadena de prefijo dada seguida de un guion bajo, por lo que `$q` se convertirá en `$fromget_q`. Te recomiendo

encarecidamente que utilices esta versión de la función cuando manejes las matrices `$_GET` y `$_POST`, o cualquier otra matriz cuyas claves las pueda controlar el usuario, ya que los usuarios maliciosos pueden enviar claves seleccionadas deliberadamente para sobrescribir nombres de variables de uso común y comprometer tu sitio web.

### compact

A veces se puede usar `compact`, el inverso de `extract`, para crear una matriz a partir de variables y sus valores. El Ejemplo 6-14 muestra cómo puede utilizar esta función.

#### Ejemplo 6-14. Uso de la función `compact`

```
<?php
    $fname      = "Doctor";
    $sname      = "Who";
    $planet     = "Gallifrey";
    $system     = "Gridlock";
    $constellation = "Kasterborous";

    $contact = compact('fname', 'sname', 'planet', 'system',
'constellation');

    print_r($contact);
?>
```

El resultado de ejecutar el Ejemplo 6-14 es el siguiente:

```
Array
(
    [fname] => Doctor
    [sname] => Who
    [planet] => Gallifrey
    [system] => Gridlock
    [constellation] => Kasterborous
)
```

Observa como `compact` requiere que los nombres de las variables se escriban entre comillas, no precedidos por el signo `$`. Esto se debe a que `compact` está buscando una lista de nombres de variables, no sus valores.

Otro uso de esta función es para depurar, cuando queremos ver rápidamente varias variables y sus valores, como en el Ejemplo 6-15.

#### Ejemplo 6-15. Uso de `compact` para ayudar en la depuración

```
<?php
    $j          = 23;
    $temp       = "Hello";
    $address    = "1 Old Street";
    $age        = 61;
```

## Aprender PHP, MySQL y JavaScript

```
print_r(compact(explode(' ', 'j temp address age')));  
?>
```

Este código utiliza la función `explode` para extraer todas las palabras de la cadena y las carga en una matriz, que luego pasa a la función `compact`, que a su vez devuelve una matriz a `print_r`, que finalmente muestra su contenido.

Si copias y pegas la línea de código `print_r`, solo necesitas alterar las variables que se nombran allí para obtener una impresión rápida de los valores de un grupo de variables. En este ejemplo, la salida es la siguiente:

```
Array  
(  
    [j] => 23  
    [temp] => Hello  
    [address] => 1 Old Street  
    [age] => 61  
)
```

## reset

Cuando la construcción `foreach...as` o la función `each` recorren una matriz, mantienen un puntero PHP interno que anota qué elemento de la matriz debe devolver a continuación. Si algunas vez necesitas que tu código vuelva al inicio de una matriz, puedes emitir un `reset`, que también devuelve el valor del primer elemento. Ejemplos de cómo utilizar esta función son los siguientes:

```
reset($fred);           // Throw away return value  
$item = reset($fred);   // Keep first element of the array in  
$item
```

## end

Al igual que con `reset`, puedes mover el puntero interno de la matriz de PHP al elemento final de un archivo con la función `end`, que también devuelve el valor del último elemento, como se puede ver en estos ejemplos:

```
end($fred);  
$item = end($fred);
```

Este capítulo concluye la introducción básica a PHP, y ahora deberías ser capaz de escribir programas bastante complejos usando las habilidades que has aprendido. En el siguiente capítulo, veremos el uso de PHP para tareas comunes y prácticas.

## Preguntas

1. ¿Cuál es la diferencia entre una matriz numérica y una matriz asociativa?
2. ¿Cuál es la principal ventaja de la palabra clave `array`?



3. ¿Cuál es la diferencia `foreach` y `each`?
4. ¿Cómo puedes crear una matriz de varias dimensiones?
5. ¿Cómo puedes determinar el número de elementos de una matriz?
6. ¿Cuál es el propósito de la función `explode`?
7. ¿Cómo se puede establecer el puntero interno de PHP en una matriz para que vuelva al primer elemento de la matriz?

Consulta "Respuestas del Capítulo 6" en la página 709 del Apéndice A para comprobar las respuestas a estas preguntas.