

## Habilidades y conceptos clave

- Crear y utilizar variables y constantes
  - Comprender los tipos de datos simples de PHP
  - Familiarizarse con algunas funciones integradas de PHP
  - Realizar operaciones aritméticas
  - Comparar y probar lógicamente las variables
  - Manejar datos enviados a través de un formulario Web
- 

**E**n el capítulo anterior se dio una introducción amigable a PHP, y dejamos que te pusieras a trabajar un poco con dos proyectos sencillos. Sin embargo, como lo comprobarás dentro de poco, PHP sirve para mucho más que llenar los espacios en blanco de una página HTML. En este capítulo aprenderás sobre variables y operadores, los dos bloques de construcción básicos para cualquier programa PHP, y los utilizarás para desarrollar programas más sofisticados. También crearás tu primer *script* interactivo, que solicita datos de entrada del usuario y responde a lo enviado. Así que, sin más preámbulos, ¡pongamos manos a la obra!

## Almacenar datos en variables

Una *variable* simplemente es un contenedor que se utiliza para almacenar información numérica y no numérica. Y como cualquier contenedor, puedes moverlo de un lugar a otro, añadirle cosas, vaciarlo en el piso y llenarlo con algo completamente diferente.

Para ampliar un poco más la analogía, así como es buena idea etiquetar todo contenedor, también debes darle un nombre a cada variable en tu programa. Como regla general, estos nombres deben tener un sentido y ser fáciles de entender. En el mundo real, esta práctica ayuda a encontrar rápidamente las cosas; en el mundo de la programación, hace que tu código sea más limpio y fácil de entender para los demás. Como alguien con la experiencia necesaria, te puedo decir que no hay nada más frustrante que pasar tres horas buscando en un montón de cajas la vajilla china preferida de mamá, sólo para descubrir que está en una caja etiquetada como “varios”, ¡junto con un hueso de plástico y bizcochos rancios!

---

En la práctica, los programadores suelen evitar los nombres de variables con acentos, diéresis y las eñes, por ser caracteres especiales del conjunto ISO-Latin-X; así se hace en los ejemplos de este libro.

PHP tiene algunas reglas sencillas para asignar nombre a las variables. Cada nombre de variable debe estar precedido por un signo de moneda (\$) y debe comenzar con una letra o un guión bajo, seguido opcionalmente por más letras, números u otros guiones bajos. Los signos de puntuación comunes, como comas, comillas o puntos no son permitidos en los nombres de las variables; tampoco los espacios en blanco. Por ejemplo, `$root`, `$_num` y `$query2` son nombres de variable válidos, mientras que `$58%`, `$1day` y `email` no son válidos.

## Asignar valores a variables

Asignar un valor a una variable en PHP es muy sencillo: se utiliza el símbolo de igual (=), que también es el operador de asignación de PHP. Este símbolo asigna el valor localizado a la derecha de la ecuación a la variable que se encuentra a la izquierda.

Para utilizar una variable en un script, simplemente se invoca su nombre en una expresión y PHP reemplaza este último por su valor cuando se ejecuta el *script*. He aquí un ejemplo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional //EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head><title /></head>
  <body>
    <?php
      // asignar valor a una variable
      $nombre = 'Juan';
    ?>
    <h2>Bienvenido al blog de <?php echo $nombre; ?></h2>
  </body>
</html>
```

En este ejemplo, se le asigna el valor 'Juan' a la variable `$nombre`. La declaración `echo` se utiliza entonces para mostrar el valor de esta variable en la página Web.

También puedes asignar a una variable el valor de otra, o el resultado de un cálculo. El siguiente ejemplo muestra ambas situaciones:

```
<?php
// asignar valor a una variable
$fecha = 2008;

// asignar el valor de la variable a otra variable
$fechaActual = $fecha;

// realizar el cálculo
$fechaAnterior = $fechaActual - 1;

// datos de salida: '2007 ha terminado. ¡Bienvenido a 2008!'
echo "$fechaAnterior ha terminado. ¡Bienvenido a $fechaActual!";
?>
```

## Pregunta al experto

**P:** ¿Es posible que un nombre de variable sea en sí una variable?

**R:** En raras situaciones, te será de utilidad asignar dinámicamente un nombre de variable, en el tiempo de ejecución. PHP te permite hacerlo al encerrar entre llaves la parte dinámica del nombre de la variable. El siguiente ejemplo ilustra este caso:

```
<?php
// definir una variable
$atributo = 'precio';

// crear una nueva variable
// su nombre surge dinámicamente
// del valor de la variable $atributo
${$atributo} = 678;

// dato de salida: 678
echo $precio;
?>
```

## Destruir variables

Para destruir una variable, pasa la variable a la función PHP llamada apropiadamente `unset()`, como en el ejemplo siguiente:

```
<?php
// asignar un valor a la variable
$carro = 'Porsche';

// mostrar el valor de la variable
// datos de salida: 'Antes de unset(), mi carro es un Porsche'
echo "Antes de unset(), mi carro es un $carro";

// destruir la variable
unset($carro);

// mostrar el valor de la variable
// esto generará un error 'undefined variable' (variable indefinida)
// datos de salida: 'después de unset(), mi carro es un '
echo "Después de unset(), mi carro es un $carro";
?>
```

**NOTA**

Tratar de utilizar o acceder a una variable que ha sido destruida con `unset()`, como en el ejemplo anterior, dará como resultado un mensaje de error *undefined variable* (variable indefinida). Este error puede o no ser visible en la página donde aparecen los datos de salida, dependiendo del nivel de reporte de errores en la configuración PHP. Para mayor información de la manera en que funcionan los mensajes de error, consulta el capítulo 10.

Como opción, es posible limpiar el contenido de la variable asignándole el valor especial PHP `NULL`. Puedes leer más acerca del tipo de dato PHP `NULL` en la siguiente sección, pero a continuación presento una revisión rápida del procedimiento para ilustrar la manera en que funciona:

```
<?php
// asignar un valor a la variable
$carro = 'Porsche';

// mostrar el valor de la variable
// datos de salida: 'Antes de unset(), mi carro es un Porsche'
echo "Antes de unset(), mi carro es un $carro";

// asignar un valor nulo a la variable
$carro = null;

// mostrar el valor de la variable
// datos de salida: 'después de unset(), mi carro es un '
echo "Después de unset(), mi carro es un $carro";
?>
```

**PRECAUCIÓN**

Los nombres de variables en PHP son sensibles a las mayúsculas. Como resultado, `$ayuda` hace referencia a una variable diferente de `$AYUDA` y `$Ayuda`. Olvidar esta sencilla regla causa mucha frustración entre los programadores novatos de PHP.

## Inspeccionar el contenido de la variable

PHP ofrece la función `var_dump()`, la cual acepta una variable y le aplica rayos X. He aquí un ejemplo:

```
<?php
// definir variables
$nombre = 'Fiona';
$edad = 28;

// mostrar el contenido de la variable
var_dump($nombre);
var_dump($edad);
?>
```

**TIP**

Existe la función `print_r()` que realiza una función semejante a `var_dump()` pero presenta menos información.

## Comprender los tipos de datos de PHP

Los valores asignados a una variable PHP pueden corresponder a diferentes *tipos de datos*, que van desde sencillas cadenas de texto y datos numéricos hasta matrices y objetos complejos. En ejemplos previos ya has visto dos de ellos en acción: cadenas de texto y números. He aquí un ejemplo integral que introduce tres tipos de datos más:

```
<?php
// Booleano
$usuarioPermitido = true;

// entero
$capacidad = 15;

// punto flotante
$temporal = 98.6;

// cadena de texto
$gato = 'Siamés';

// nulo
$lugar = null;
?>
```

- Los *booleanos* son los tipos de datos más sencillos de PHP. Como un conmutador que sólo tiene dos estados: encendido y apagado, consiste en un solo valor que puede ser establecido como 1 (`true` [verdadero]) o 0 (`false` [falso]). En el ejemplo anterior `$usuarioPermitido` es una variable booleana establecida como verdadera (`true`).
- PHP también admite dos tipos de datos numéricos: *enteros* y *valores de punto flotante*. Los valores de punto flotante (también conocidos como *flotantes* o *dobles*) son números decimales o fracciones, mientras que los enteros son números naturales. Ambos pueden ser menores que, mayores que o iguales a cero. En el ejemplo, la variable `$capacidad` contiene un valor entero, mientras que la variable `$temporal` contiene un valor de punto flotante.
- Para datos diferentes a los numéricos, PHP ofrece el tipo de dato cadena de caracteres (*string*), que puede almacenar letras, números y caracteres especiales. Las cadenas de caracteres deben ir encerradas entre comillas dobles o sencillas. En el ejemplo anterior, `$gato` es una variable de cadena de texto que contiene el valor `'Siamés'`.

- También tenemos el tipo de dato NULL (nulo), que es un tipo de dato “especial” introducido por primera vez en PHP 4. NULL es utilizado en PHP para representar variables “vacías”; una variable de tipo NULL es una variable sin datos. En el ejemplo anterior, \$lugar es una variable NULL.

## Pregunta al experto

**P:** ¿PHP acepta números escritos en hexadecimal, octal o notación científica?

**R:** Sí, sí y sí. He aquí algunos ejemplos:

```
<?php
// 8, especificado como un valor octal
$a = 010;

// 1500, especificado como un valor hexadecimal
$b = 0x5dc;

// 690, en notación científica
$c = 6.9E+2;
?>
```

### **PRECAUCIÓN**

Muchos desarrolladores novatos en PHP creen erróneamente que al asignar a una variable el valor de una cadena de caracteres vacía (") automáticamente queda vacía. Esto no es cierto, porque PHP no considera equivalentes los valores de una cadena de texto vacía y un tipo de datos NULL. Para eliminar por completo el contenido de una variable, siempre debe declararse el tipo de datos NULL.

## Establecer y verificar el tipo de datos de la variable

Contrariamente a otros lenguajes de programación, donde el tipo de dato el programador debe definir explícitamente la variable, PHP determina de manera automática el tipo de variable por el contenido que almacena. En caso de que el contenido de la variable cambie durante la duración del script, el lenguaje establecerá automáticamente el nuevo tipo de dato apropiado para la variable, de acuerdo con el cambio.

He aquí un ejemplo que ilustra este *malabarismo con los tipos de datos*:

```
<?php
// define una variable cadena de caracteres
$soy = 'Sara';

// datos de salida: 'cadena de caracteres'
echo gettype($soy);
```

```
// asigna un nuevo valor entero a la variable
$soy = 99.8;

// datos de salida: 'doble'
echo gettype($soy);

// destruye la variable
unset($soy);

// datos de salida: 'NULL'
echo gettype($soy);
?>
```

Este ejemplo presenta por primera vez el operador PHP `gettype()`, que es una herramienta útil y ligera que se utiliza para averiguar el tipo de una variable. Como lo muestran los datos de salida del script, la variable `$soy` inicia como una cadena de caracteres, con el valor 'Sara'. Después se le asigna como valor el número 99.8, con lo que se convierte automáticamente en una variable de punto flotante. A continuación, la variable se invalida con el método `unset()`, el cual borra su valor y la transforma en una variable `NULL`. PHP es la mano invisible detrás de todas estas transformaciones, restableciendo internamente el tipo de dato de la variable `$soy` de una cadena de texto a un número de punto flotante y de ahí a un valor nulo.

Sin embargo, esto no significa que estés por completo a merced de PHP; es posible establecer específicamente el tipo de variable PHP al *convertir* la variable en un tipo específico antes de utilizarla. La conversión es una técnica de uso común para los programadores de Java; para utilizarla, simplemente especifica entre paréntesis, y del lado derecho de la ecuación, el tipo de dato que deseas asignar a una variable. Considera el siguiente ejemplo, que ilustra la conversión de un valor de punto flotante a un entero:

```
<?php
// define una variable de punto flotante
$velocidad = 501.789;

// conversión a entero
$nuevaVelocidad = (integer)$velocidad;

// datos de salida: 501
echo $nuevaVelocidad;
?>
```

Además de la función `gettype()`, PHP cuenta con otras más especializadas para probar si una variable corresponde a un tipo de datos específico. La tabla 2-1 tiene la lista de estas funciones.

### **TIP**

¿Recuerdas la función `var_dump()` que vimos en la sección anterior? Si miras con cuidado los datos de salida que genera, notarás que además de decirte lo que contiene la variable, también muestra el tipo de datos correspondiente.

Función	Propósito
is_bool()	Prueba si la variable contiene un valor booleano
is_numeric()	Prueba si la variable contiene un valor numérico
is_int()	Prueba si la variable contiene un valor entero
is_float()	Prueba si la variable contiene un valor de punto flotante
is_string()	Prueba si la variable contiene un valor de cadena de texto
is_null()	Prueba si la variable contiene un valor NULL
is_array()	Prueba si el valor que contiene la variable es una matriz
is_object()	Prueba si el valor que contiene la variable es un objeto

**Tabla 2-1** Funciones de PHP para probar los tipos de dato de las variables

## Usar constantes

Hasta ahora el capítulo se ha concentrado en las variables, que son útiles para almacenar y cambiar valores durante el tiempo de vida del script PHP. Pero, ¿qué sucede si necesitas almacenar un valor fijo, que permanezca estático durante el curso del script? Bien, en esos casos se debe utilizar una *constante*.

Como su nombre lo sugiere, las constantes son contenedores de PHP para valores que permanecen constantes y que nunca cambian. Por lo regular se utilizan para valores bien conocidos de antemano y que son utilizados, sin cambio alguno, en diferentes lugares de la aplicación. Buenos candidatos para constantes son los niveles de depuración y registro, los números de versión, las marcas de configuración y las fórmulas.

Las constantes se definen en PHP con la función `define()`, la cual acepta dos argumentos: el nombre de la constante y su valor. Los nombres de las constantes deben seguir las mismas reglas que las variables, con una sola excepción: el prefijo `$` no se requiere para los nombres de constantes.

He aquí un ejemplo para definir y utilizar una constante dentro de un script:

```
<?php
// define constantes
define('PROGRAMA', 'The Matrix');
define('VERSION', 11.7);

// usar constantes
// datos de salida: 'Bienvenido a The Matrix (versión 11.7)'
echo 'Bienvenido a ' . PROGRAMA . ' (versión ' . VERSION . ')';
?>
```



**NOTA**

Por convención, los nombres de constantes se escriben en mayúsculas; esto se hace con el fin de identificarlos y diferenciarlos rápidamente de las variables “regulares” en un script.

## Manipular variables con operadores

Por sí solas, las variables sólo son contenedores de información. Para realizar algo útil con ellas necesitas *operadores*. Los procesadores son símbolos que le indican al procesador PHP que realice ciertas acciones. Por ejemplo, el símbolo de suma (+) le dice a PHP que sume dos variables o valores, mientras que el símbolo mayor que (>) es un operador que le dice a PHP que compare dos valores.

PHP da soporte a más de 50 de estos operadores, que van desde los destinados a operaciones aritméticas hasta operadores para comparaciones lógicas y cálculos complejos. En esta sección se abordan los operadores de uso más común.

### Pregunta al experto

**P:** ¿Cuándo debo usar una variable y cuándo una constante?

**R:** Las variables son para almacenamiento temporal; utilízalas para valores que tal vez cambiarán a lo largo del script. Las constantes son un poco más permanentes; utilízalas para valores que tal vez se mantendrán fijos y a los que se hacen múltiples referencias dentro del script. A diferencia de las variables, las constantes no pueden destruirse y no pueden generarse sus nombres dinámicamente.

## Realizar operaciones aritméticas

PHP da soporte a todas las operaciones aritméticas estándar, como se muestra en la lista de operadores de la tabla 2-2.

Operador	Descripción
+	Suma
-	Resta
*	Multipliación
/	Divide y regresa el cociente
%	Divide y regresa el residuo

**Tabla 2-2** Operadores aritméticos comunes

He aquí un ejemplo que muestra estos operadores en acción:

```
<?php
// define variables
$x = 10;
$y = 5;
$z = 3;

// suma
$suma = $x + $y;
echo "$x + $y = $suma\n";

// resta
$resta = $x - $y;
echo "$x - $y = $resta\n";

// multiplica
$producto = $x * $y;
echo "$x * $y = $producto\n";

// divide y obtiene el cociente
$cociente = $x / $y;
echo "$x / $y = $cociente\n";

// divide y obtiene el módulo
$modulo = $x % $y;
echo "$x % $y = $modulo\n";
?>
```

## Pregunta al experto

**P:** ¿Hay algún límite para la longitud de los enteros?

**R:** Sí lo hay, pero es muy alto: 2147483647. Este límite está definido por PHP en su constante `PHP_INT_MAX`, así que puedes comprobarlo tú mismo en cualquier momento.

## Unir cadenas de texto

Para combinar cadenas de texto, utiliza el operador de unión de PHP, el cual resulta ser un punto (.). El siguiente ejemplo lo ilustra:

```
<?php
// define variables
$pais = 'Inglaterra';
$ciudad = 'Londres';
```

```
// combina ambas en una sola línea
// datos de salida: 'Bienvenido a Londres, la ciudad más fría de toda Inglaterra'
echo 'Bienvenido a ' . $ciudad . ', la ciudad más fría de toda ' . $pais;
?>
```

## Comparar variables

PHP te permite comparar una variable o un valor con otro mediante su amplia variedad de operadores de comparación, presentados en la tabla 2-3.

He aquí un ejemplo para mostrar estos operadores en acción:

```
<?php
// define variables
$p = 10;
$q = 11;
$r = 11.3;
$s = 11;

// prueba si $q es mayor que $p
// regresa el valor de verdadero (true)
echo ($q > $p);

// prueba si $q es menor que $p
// regresa el valor de falso (false)
echo ($q < $p);

// prueba si $q es mayor que o igual a $s
// regresa el valor de verdadero (true)
echo ($q >= $s);

// prueba si $r es menor o igual que $s
// regresa el valor de falso (false)
echo ($r <= $s);
```

Operador	Descripción
==	Igual a
!=	Diferente de
>	Mayor que
>=	Mayor que o igual a
<	Menor que
<=	Menor que o igual a
===	Igual a y del mismo tipo

**Tabla 2-3** Operadores de comparación comunes

```
// prueba si $q es igual a $s
// regresa el valor de verdadero (true)
echo ($q == $s);

// prueba si $q es igual a $r
// regresa el valor de falso (false)
echo ($q == $r);
?>
```

Mención especial merece aquí el operador `===`, excluido del ejemplo anterior. Este operador permite realizar una comparación estricta entre variables: sólo regresa el valor verdadero (`true`) si las dos variables o valores comparados tienen la misma información y son del mismo tipo de datos. De esta manera, en el ejemplo siguiente la comparación entre las variables `$bool` y `$num` regresará el valor verdadero (`true`) cuando se les compare con el operador `==`, pero falso (`false`) cuando se les compare con `===`:

```
<?php
// define variables de dos tipos
// pero con el mismo valor
$bool = (boolean) 1;
$num = (integer) 1;

// regresa el valor de verdadero (true)
echo ($bool == $num);

// regresa el valor de falso (false)
echo ($bool === $num);
?>
```

## Realizar pruebas lógicas

Cuando se construyen expresiones condicionales complejas (tema que se abordará con detalle en el capítulo 3), constantemente te encontrarás con situaciones en las que es necesario combinar una o más pruebas lógicas. Los tres operadores lógicos más usados en PHP, listados en la tabla 2-4, están pensados específicamente para esas situaciones.

Operador	Descripción
<code>&amp;&amp;</code>	Y (AND)
<code>  </code>	O (OR)
<code>!</code>	NO (NOT)

**Tabla 2-4** Operadores lógicos comunes

Los operadores lógicos en realidad muestran su poder cuando se combinan con pruebas condicionales; el siguiente ejemplo es sólo para ilustrar esta gran capacidad; en el capítulo 3 encontrarás mejor material para entretenerte.

```
<?php
// define variables
$precio = 100;
$tamano = 18;

// prueba lógica Y (AND)
// regresa el valor de verdadero (true) si ambas comparaciones son verdaderas
// en este caso regresa el valor de verdadero (true)
echo ($precio > 50 && $tamano < 25);

// prueba lógica O (OR)
// regresa el valor de verdadero (true) si cualquiera de las
comparaciones es verdadera
// en este caso regresa el valor de falso (false)
echo ($precio > 150 || $tamano > 75);

// prueba lógica NO (NOT)
// invierte la prueba lógica
// en este caso regresa el valor de falso (false)
echo !($tamano > 10);
?>
```

## Otros operadores útiles

Hay unos cuantos operadores más que suelen ser útiles durante el desarrollo de PHP. Primero, el operador de asignación de suma, representado por el símbolo `+=`, permite sumar y asignar un nuevo valor a la variable simultáneamente. El siguiente ejemplo lo ilustra:

```
<?php
// define variable
$cuenta = 7;

// suma 2 y asigna el valor de la suma a la variable
$cuenta += 2;

// dato de salida: 9
echo $cuenta;
?>
```

En el ejemplo anterior, la expresión `$cuenta += 7` es equivalente a la expresión `$cuenta + 2`, una operación de suma seguida por la asignación del resultado a la misma variable. En la misma línea de acción, existen operadores para otras asignaciones matemáticas y cadenas de texto. La tabla 2-5 presenta la lista.

Operador	Descripción
+=	Suma y asigna
-=	Resta y asigna
*=	Multiplica y asigna
/=	Divide y asigna el cociente
%=	Divide y asigna el residuo
.=	Concatena y asigna (solamente cadenas de texto)

**Tabla 2-5** Operadores de asignación comunes

He aquí algunos ejemplos de estas acciones:

```
<?php
// define variables
$cuenta = 7;
$edad = 60;
$saludo = 'Bien';

// resta 2 y reasigna el nuevo valor a la variable
// equivalente a $cuenta = $cuenta - 2
// dato de salida: 5
$cuenta -= 2;
echo $cuenta;

// divide entre 5 y reasigna el nuevo valor a la variable
// equivalente a $edad = $edad / 5
// dato de salida: 12
$edad /= 5;
echo $edad;

// añade una nueva cadena de texto y reasigna el nuevo valor a la
variable
// equivalente a $saludo = $saludo . 'venidos'
// dato de salida: 'Bienvenidos'
$saludo .= 'venidos';
echo $saludo;
?>
```

Más adelante, en este libro, también encontrarás los operadores de autoincremento y autodecremento, representados por los símbolos ++ y --, respectivamente.

Estos operadores suman 1 o restan 1 automáticamente a la variable en que se aplican. He aquí un ejemplo:

```
<?php
// define variable
$cuenta = 19;

// incremento
$cuenta++;

// dato de salida: 20
echo $cuenta;

// ahora el decremento
$cuenta--;

// dato de salida: 19
echo $cuenta;
?>
```

Estos operadores suelen encontrarse en contadores reiterativos, otro tema que abordaremos con detalle en el capítulo 3.

## Comprender la precedencia de los operadores

Recordando las clases de matemáticas, tal vez te enseñaron el CODMSR, acrónimo que especifica el orden en el que una calculadora o una computadora realiza una secuencia de operaciones matemáticas: Corchete, Orden, División, Multiplicación, Suma y Resta. Pues bien, PHP sigue un conjunto de reglas similares para determinar cuáles operadores tienen precedencia sobre otros, y aprender estas reglas te ahorrará incontables horas de frustración depurando un cálculo que parece bien estructurado y en el que, sin embargo, algo anda mal, ¡porque siempre regresa un resultado erróneo!

La siguiente lista (una versión reducida de una mucho más larga que forma parte del manual de PHP) presenta las reglas de precedencia más importantes. Los operadores en el mismo nivel tienen precedencia equivalente.

- ++ --
- !
- \* / %
- + - .
- < <= > >=

- == != === !==
- &&
- ||
- = += -= \*= /= .= %= &= |= ^=

## Pregunta al experto

**P:** Las reglas de precedencia de PHP son difíciles de recordar. ¿Existe alguna otra manera de indicar a PHP el orden en que quiero que se realicen los cálculos?

**R:** Sí. Los paréntesis siempre tienen la más alta precedencia, por lo que encerrar una expresión entre paréntesis obligará a PHP a evaluarlos primero. Cuando utilices paréntesis anidados, recuerda que la evaluación comienza con el último conjunto de paréntesis de izquierda a derecha y sigue el orden de dentro hacia afuera (como si se quitaran capas de una cebolla desde adentro). A manera de ejemplo, considera la expresión `((4 * 8) - 2) / 10`, que da como resultado 3 con paréntesis y `31.8` sin ellos.

## Prueba esto 2-1

## Construir un convertidor dólar-euro

Ahora tomemos un breve descanso de toda esta teoría e intentemos aplicar algo de ella a un proyecto práctico y real: un convertidor monetario de dólares a euros. En este proyecto se aplicarán algunos conocimientos que has aprendido en secciones anteriores sobre variables, constantes y operadores aritméticos; ¡también será de utilidad la próxima vez que tomes tu avión para pasar las vacaciones en Europa!

He aquí el código (*convertidor.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 2-1: Convertidor Monetario USD/EUR</title>
  </head>
  <body>
    <h2>Proyecto 2-1: Convertidor Monetario USD/EUR</h2>
    <?php
      // define tasa de cambio
```

(continúa)



```
// 1.00 USD = 0.70 EUR
define ('TASA_DE_CAMBIO', 0.70);

// define la cantidad de dólares
$dolares = 150;

// realiza la conversión y presenta el resultado
$euros = $dolares * TASA_DE_CAMBIO;
echo "$dolares USD americanos son equivalentes a: $euros EUR";
?>
</body>
</html>
```

Si has seguido con cuidado los ejemplos, debes entender este script con gran facilidad. Comienza por definir una constante llamada `TASA_DE_CAMBIO` que (¡sorpresa!) almacena la tasa de cambio entre el dólar y el euro (aquí damos por hecho que un dólar equivale a 0.70 euros). A continuación, define una variable llamada `$dolares`, que almacena el monto que será convertido a euros, y después realiza una operación aritmética utilizando el operador `*` sobre la variable `$dolares` y la constante `TASA_DE_CAMBIO` para obtener el número equivalente en euros. El resultado se almacena en una nueva variable llamada `$euros`, misma que es presentada en la página Web.

La figura 2-1 ilustra la manera en que aparecen los datos de salida.

Para convertir una cantidad diferente de dólares, simplemente cambia el valor de la variable `$dolares`. ¡Vamos, inténtalo y compruébalo tú mismo!



**Figura 2-1** Los datos de salida del convertidor dólares-euros

## Manejar datos de entrada para formularios

Hasta ahora, todos los ejemplos que has visto tienen sus variables claramente definidas al inicio del script. Sin embargo, a medida que tus scripts PHP se hagan más complejos, esta feliz situación cambiará y tendrás que aprender a interactuar con datos de entrada que proporciona el usuario. La fuente más común para transmitir estos datos es un formulario Web, y PHP cuenta con un mecanismo sencillo para recuperar información enviada en estos formularios.

Como ejemplo, considera el siguiente formulario Web simple (*lista.html*), con el que se selecciona una marca de automóvil y se ingresa el color deseado:

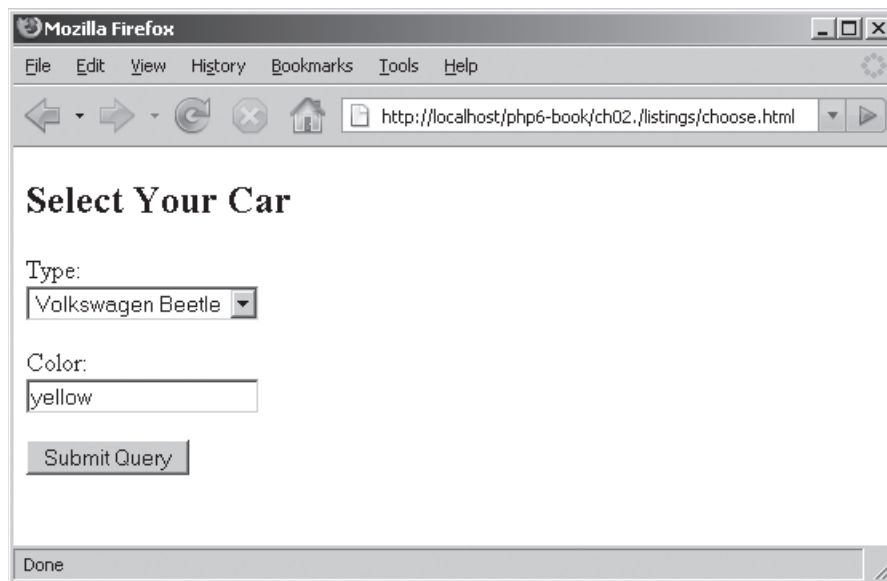
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml11-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head><title /></head>
  <body>
    <h2>Selecciona Tu Automóvil</h2>
    <form method="post" action="carro.php">
      Tipo: <br />
      <select name="tipo">
        <option value="Porsche 911">Porsche 911</option>
        <option value="Volkswagen Beetle">Volkswagen Beetle</option>
        <option value="Ford Taurus">Ford Taurus</option>
      </select><p />
      Color: <br />
      <input type="text" name="txtColor" /> <p />
      <input type="submit" />
    </form>
  </body>
</html>
```

Hasta aquí el formulario es muy sencillo: tiene una lista de selección y un recuadro de texto. La figura 2-2 muestra cómo aparece en el explorador Web.

Pon atención al atributo 'action' del formulario Web: hace referencia al script PHP llamado *carro.php*. Éste es el script que recibe los datos del formulario una vez que éste ha sido enviado. También debes prestar atención al atributo 'method' del formulario, el cual especifica que el envío de los datos será a través del método POST.

Con estos dos datos bien comprendidos, veamos ahora el script *carro.php*:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml11-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head><title /></head>
  <body>
    <h2>¡Éxito!</h2>
```



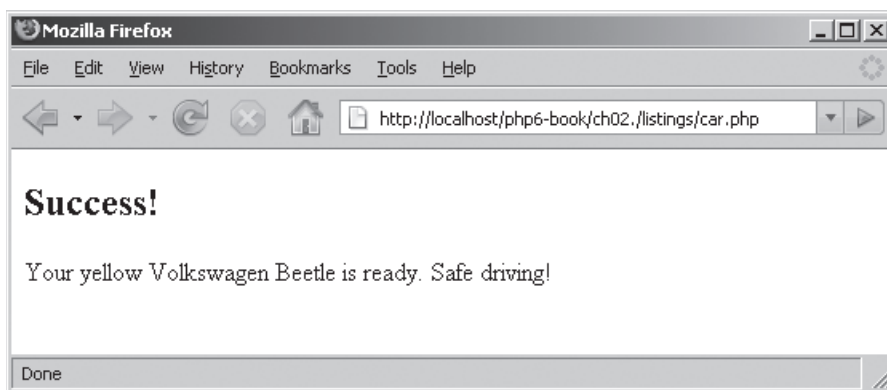
**Figura 2-2** Un formulario sencillo

```
<?php
// obtiene los datos de entrada del formulario
$tipo = $_POST['tipo'];
$color = $_POST['txtColor'];

// utiliza los datos de entrada del formulario
echo "Tu $tipo $txtColor está listo. ¡Maneja con cuidado!";
?>
</body>
</html>
```

¿Qué sucede aquí? Bien, cada vez que un formulario es enviado a un script PHP con el método POST, las variables internas del formulario y sus respectivos valores están disponibles para el script PHP a través de un contenedor de variables especial llamado `$_POST`. Accesar al valor introducido en un campo particular del formulario se convierte en una simple referencia a `$_POST` con el correspondiente nombre del campo, como se aprecia en el script anterior.

Considera, por ejemplo, la tarea de acceder al color escrito por el usuario en el formulario Web. En el código del formulario se puede ver que el campo para la inserción de datos designado para esta información lleva el nombre `'txtColor'`. Por lo tanto, dentro del script PHP, el valor ingresado en este campo de texto puede ser accesado utilizando la sintaxis `$_POST['txtColor']`.



**Figura 2-3** El resultado de enviar el formulario

Este valor puede ser utilizado de la manera convencional: imprimirse en una página Web, asignarse a otra variable o manipularse con los operadores presentados en las secciones anteriores.

La figura 2-3 muestra el resultado enviado por el formulario.

PHP también contempla los casos en que el formulario Web envía datos utilizando el método GET, en lugar de POST: los datos de entrada del formulario enviados con el método GET encuentran su equivalente en el contenedor de variables `$_GET`, el cual puede utilizarse al hacer referencia a `$_GET` en lugar de `$_POST`.

## Pregunta al experto

**P:** Entendí en las secciones anteriores los aspectos de las variables y la manera en que funcionan, pero `$_GET` y `$_POST` no siguen las mismas reglas. ¿Qué sucede aquí?

**R:** Las variables `$_GET` y `$_POST` son diferentes a los tipos de variables numéricas y las cadenas de texto que hemos visto en este capítulo. Son un tipo de variables más complejas llamadas *matrices*, las cuales pueden contener más de un valor al mismo tiempo y también siguen reglas diferentes para almacenar y acceder a valores dentro de ellas. Las matrices se abordarán ampliamente en el capítulo 4; en ese punto todo lo que has leído sobre `$_GET` y `$_POST` cobrará sentido.

## Prueba esto 2-2 Construir un muestrario HTML interactivo de colores

Ahora que has aprendido a acceder datos de entrada del formulario mediante un script PHP, elaboremos una aplicación que muestre esta característica de manera más específica. En este proyecto, construirás un muestrario de colores HTML, una herramienta que te permitirá ingresar valores RGB de colores y presentará una muestra del tono correspondiente en el explorador Web. Los valores RGB de los colores se introducirán en un formulario y PHP los procesará, con lo que crearemos un ejemplo aplicable a la realidad con los conocimientos que has adquirido en las secciones anteriores.

Primero el formulario Web (*color.html*):

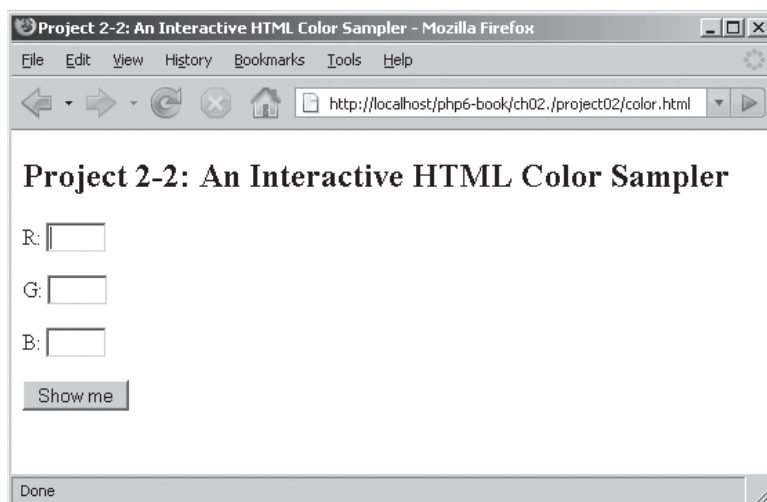
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 2-2: Muestrario HTML Interactivo de Colores</title>
  </head>
  <body>
    <h2>Proyecto 2-2: Muestrario HTML Interactivo de Colores</h2>
    <form method="get" action="muestra.php">
      R: <input type="text" name="r" size="3" /> <p />
      G: <input type="text" name="g" size="3" /> <p />
      B: <input type="text" name="b" size="3" /> <p />
      <input type="submit" value="Muéstrame" />
    </form>
  </body>
</html>
```

Hasta ahora es muy convencional: un formulario Web con tres campos de datos de entrada, cada uno de ellos asignado a un componente específico de color (rojo [R], verde [G] y azul [B]). Advierte que este formulario envía sus datos a un script PHP llamado *muestra.php*, y utiliza el método GET (para variar).

La figura 2-4 muestra cómo se ve en el explorador Web.

A continuación, crearemos el script PHP que acepta los datos de entrada y los utiliza para mostrar los colores (*muestra.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 2-2: Muestrario HTML Interactivo de Colores</title>
  </head>
  <body>
```



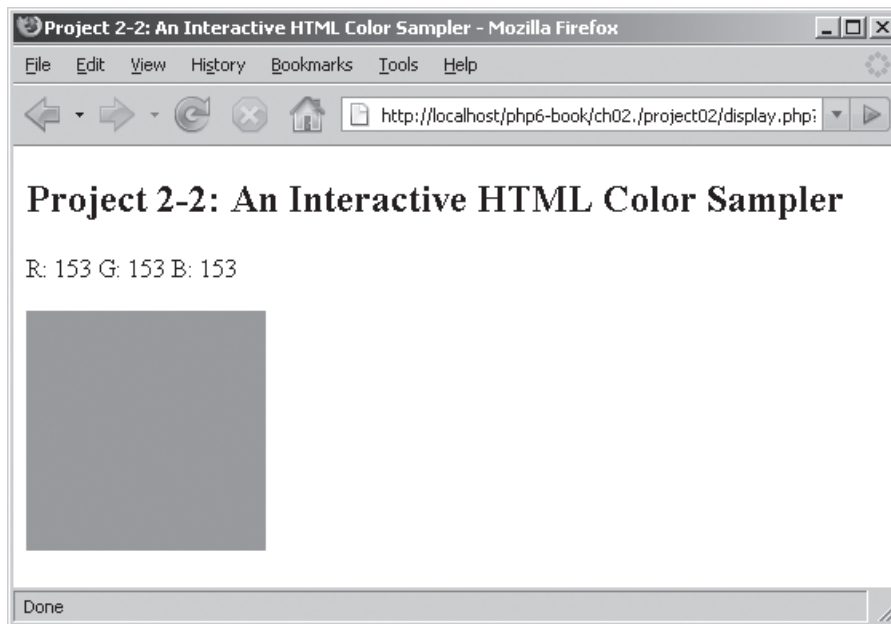
**Figura 2-4** Un formulario Web con campos para los valores RGB de color

```
<h2>Proyecto 2-2: Muestrario HTML Interactivo de Colores</h2>
<?php
// obtiene los valores de entrada
$r = $_GET['r'];
$g = $_GET['g'];
$b = $_GET['b'];

// genera la cadena de caracteres RGB para los datos de entrada
$rgb = $r . ',' . $g . ',' . $b;
?>
  R: <?php echo $r; ?>
  G: <?php echo $g; ?>
  B: <?php echo $b; ?>
<p />
<div style="width:150px; height: 150px;
background-color: rgb(<?php echo $rgb; ?>)" />
</body>
</html>
```

Los datos insertados en el formulario quedan accesibles a través de la matriz `$_GET` (advierte que es `$_GET` porque el método utilizado en el formulario para enviar los datos fue GET). El script divide esta información en tres variables: `$r`, `$g` y `$b`. Estas variables son unidas en una sola línea utilizando el operador de unión de PHP (¿lo recuerdas?). Esta línea RGB es utilizada después para establecer el color de fondo para un elemento `<div>` en la parte inferior del documento HTML.

(continúa)



**Figura 2-5** La muestra del color que se despliega al enviar el formulario

¿El resultado? Cuando el explorador Web construye la página, verás un bloque de  $150 \times 150$  píxeles relleno con el color sólido correspondiente a los valores RGB seleccionados. Y si regresas al formulario e insertas un nuevo conjunto de código RGB, el color cambiará de acuerdo con esos valores... ¡De ahí lo “interactivo” que ostenta el título del proyecto!

La figura 2-5 muestra lo que sucede cuando se insertan los valores RGB 153, 153, 153 (un tono gris claro, porque este libro está impreso en blanco y negro).

La tabla 2-6 contiene algunas combinaciones de color RGB para que las pruebes.

Color	R	G	B
Rosa	250	75	200
Naranja	255	105	0
Amarillo	255	255	0
Verde	75	200	60
Café	100	75	25

**Tabla 2-6** Muestra de combinaciones de color RGB

## Resumen

Tal vez te tomó un poco más de tiempo terminar este capítulo en comparación con el anterior, pero al final tus conocimientos sobre la construcción básica de PHP debieron aumentar considerablemente. El capítulo inició presentando variables y constantes PHP, explicando la manera de darles nombre, asignarles valores, utilizarlas en un script y destruirlas cuando ya no se requieren. Te presentó una breve introducción a los tipos de datos simples de PHP y después te llevó en un viaje relámpago por los operadores aritméticos, las cadenas de texto, de comparación y lógicos, utilizando ejemplos comentados para explicar la manera en que cada uno de estos operadores puede ser utilizado para manipular y modificar el contenido de las variables.

Con estas bases bien asentadas, el capítulo prosiguió con una explicación sobre cómo procesar los datos de entrada de un formulario con PHP, tarea común que realizarás una y otra vez a lo largo de este curso y, en realidad, a través de tu carrera como desarrollador de PHP. Por último, dos proyectos mostraron la manera práctica de aplicar estos conocimientos: uno mostró el uso de operadores para realizar cálculos con las variables y el otro mostró lo fácil que es crear una aplicación interactiva PHP utilizando datos de entrada provenientes de un formulario.

El siguiente capítulo estará basado en todo lo que has aprendido hasta ahora, explicará cómo puedes añadir inteligencia a tus scripts PHP a través del uso de pruebas condicionales y te mostrará cómo los constructores de bucles de PHP pueden ayudarte a realizar acciones repetitivas. Hasta entonces, ocupa algo de tiempo revisando los siguientes vínculos Web del manual PHP, que ofrecen información más detallada de los temas abordados en este capítulo:

- Tipos de datos PHP, en [www.php.net/manual/en/language.types.php](http://www.php.net/manual/en/language.types.php)
- Malabarismos con los tipos de datos y su conversión en PHP, [www.php.net/manual/en/language.types.type-juggling.php](http://www.php.net/manual/en/language.types.type-juggling.php)
- Tablas de comparaciones de tipos de datos PHP, en [www.php.net/manual/en/types.comparisons.php](http://www.php.net/manual/en/types.comparisons.php)
- Conocimientos básicos sobre variables, en [www.php.net/manual/en/language.variables.php](http://www.php.net/manual/en/language.variables.php)
- Operadores PHP y precedencia de operadores, en [www.php.net/manual/en/language.operators.php](http://www.php.net/manual/en/language.operators.php)
- Accesar datos de un formulario con PHP, en [www.php.net/manual/en/language.variables.external.php](http://www.php.net/manual/en/language.variables.external.php)



## ✓ Autoexamen Capítulo 2

1. La función PHP para detectar el tipo de variables es: \_\_\_\_\_
2. Identifica cuáles de los siguientes son nombres de variables no válidos: \$24, \$SOYYO, \$\_error, \$^b, \${\$var}, \$yA\_K
3. Escribe una declaración PHP para crear el valor de una constante que almacene el nombre de tu helado favorito.
4. Escribe un script PHP para inicializar una variable y luego incrementar su valor de 3 en 3.
5. Marca como verdaderas o falsas las siguientes declaraciones:
  - A La función unset () borra una variable y la elimina del espacio de variables del programa.
  - B Las expresiones PHP \$c = '' y \$c = null son equivalentes.
  - C El resultado del cálculo (56 - 1 \* 36 % 7) es 6.
  - D El operador == compara el valor y el tipo de variable.
  - E El operador lógico O tiene una precedencia superior que el operador lógico Y.
  - F La función is\_numeric () regresa el valor true (verdadero) si se aplica a un valor de punto flotante.
  - G Convertir un número de punto flotante a entero siempre da como resultado un valor redondeado.
  - H Los elementos tipo 'hidden' del formulario se excluyen de \$\_POST y \$\_GET.

6. ¿Cuáles son los valores de \$x y ABC al finalizar el siguiente script?

```
<?php
$x = 89;
define ('ABC', $x+1);
$x += ABC;
?>
```

7. ¿Cuáles son los posibles datos de salida del siguiente script PHP?

```
<?php
$boolean = (integer) true;
$numero = 1;
echo (integer) ($boolean === $numero);
?>
```

- 8.** ¿Cuáles son los posibles datos de salida del siguiente script PHP?

```
<?php
define ('NUM', '7');
$a = NUM;
echo gettype ($a);
?>
```

- 9.** Reescribe el código de Prueba esto 2-1, de tal manera que el usuario proporcione la tasa de cambio y la cantidad convertida, a través de un formulario Web.
- 10.** Escribe un script PHP que acepte el valor de la temperatura en grados Celsius (C) mediante un formulario Web y que los convierta a la escala de grados Fahrenheit (F). La fórmula que debe usarse para la conversión es:  $F = (9/5) * C + 32$ .
- 11.** Escribe un script PHP que muestre los valores insertados en un formulario Web que contenga:
- Un campo de texto.
  - Un área de texto.
  - Un campo oculto.
  - Un campo de contraseña.
  - Una lista de selección.
  - Dos botones de opción.
  - Dos casillas de verificación.



# Capítulo 3

## Controlar el flujo del programa

## Habilidades y conceptos clave

- Aprender a utilizar declaraciones condicionales como `if-else` y `switch-case`
  - Automatizar tareas repetitivas con los bucles `while`, `do-while` y `for`
  - Ganar experiencia con las funciones numéricas y de cadenas de texto integradas en PHP
- 

Los programas PHP que viste en el capítulo anterior fueron muy convencionales: aceptaban uno o más valores de entrada, realizaban cálculos o comparaciones con ellos y regresaban un resultado. Sin embargo, en la realidad los programas PHP no son tan sencillos: muchos necesitarán tomar decisiones complejas y ejecutar diferentes operaciones durante su ejecución, de acuerdo con las condiciones específicas marcadas por el programador.

En este capítulo aprenderás a crear programas PHP que son más “inteligentes” y pueden realizar diferentes acciones de acuerdo con los resultados obtenidos en pruebas lógicas y comparativas. También aprenderás a volver automáticas acciones repetitivas utilizando bucles y aprenderás más sobre las funciones integradas de PHP para trabajar con cadenas de texto y números. Para asegurar que puedes aplicar estos conocimientos en la realidad, este capítulo también te permite aplicar tus conocimientos recién adquiridos en cuatro proyectos prácticos.

## Escribir declaraciones condicionales sencillas

Además de almacenar y recuperar valores en variables, PHP también permite que los programadores evalúen diferentes condiciones durante el curso del programa y que tomen decisiones basándose en el resultado verdadero o falso de la evaluación. Estas condiciones y las acciones asociadas con ellas se expresan mediante un constructor de programación llamado *declaración condicional*. PHP da soporte a diferentes tipos de declaraciones condicionales, cada una de ellas diseñada para un uso específico.

### La declaración `if`

La declaración condicional más sencilla de PHP es `if`. Funciona de manera muy semejante a su correspondiente en el lenguaje común: “**si** (if) sucede X, haz Y”. He aquí un sencillo ejemplo; contiene una declaración condicional que verifica si el valor de la variable `$numero` es menor que 0 y envía un mensaje de notificación en caso positivo.

```
<?php
//si el número es menor que cero
//presenta el mensaje
$numero = -88;
if ($numero < 0) {
    echo 'Este número es negativo';
}
?>
```

De esta manera, la clave para la declaración `if` es la condición que habrá de evaluarse, que siempre debe ir encerrada entre paréntesis. Si la condición evaluada es verdadera, se ejecuta el código encerrado entre las llaves (`{ }`); en caso de que la evaluación resulte falsa, se omite el código entre las llaves. Esta prueba verdadero/falso se realiza utilizando los operadores de comparación PHP, que conociste en el capítulo anterior; la tabla 3-1 hace una recapitulación rápida de ellos.

## La declaración `if-else`

La declaración `if` es muy básica; sólo te permite definir lo que sucede cuando la condición especificada se evalúa como verdadera. Pero PHP también ofrece la declaración `if-else`, una versión mejorada del constructor `if` que te permite definir un conjunto opcional de acciones que el programa debe ejecutar cuando la condición especificada se evalúa como falsa. Por lo general, el uso de esta declaración trae como resultado código más compacto y legible, porque te permite combinar dos acciones en un solo bloque de código unificado. En lenguaje común, esta declaración podría leerse como: “si sucede X, haz Y; de otra manera, haz Z”.

Operador	Descripción
<code>==</code>	Igual a
<code>!=</code>	Diferente de
<code>&gt;</code>	Mayor que
<code>&gt;=</code>	Mayor que o igual a
<code>&lt;</code>	Menor que
<code>&lt;=</code>	Menor que o igual a
<code>===</code>	Igual a y del mismo tipo

**Tabla 3-1** Operadores de comparación comunes

Como ejemplo, examina esta revisión del ejemplo anterior:

```
<?php
//cambia el mensaje dependiendo de
//si el número es menor que cero o no lo es
$numero = -88;
if ($numero < 0) {
    echo 'Este número es negativo';
} else {
    echo 'Este número es positivo o igual a cero';
}
?>
```

Aquí, la declaración `if-else` se utiliza para manejar dos posibles resultados: un número menor que cero y todos los demás números. Para verlo en acción, intenta ejecutar el script una vez tal y como aparece aquí y vuelve a ejecutarlo después de cambiar el valor de la variable `$numero` por uno positivo.

## Pregunta al experto

**P:** ¿Existe una manera más compacta de escribir la declaración `if-else`?

**R:** Sí. Requiere un pequeño truco llamado *operador ternario*. Este operador, representado por el signo de cierre de interrogación (`?`), te permite representar la declaración condicional `if-else` en una sola y compacta línea de código. Para verlo en acción, examina los siguientes scripts; ambos son equivalentes:

El bloque if-else estándar	El bloque equivalente utilizando el operador ternario
<pre>&lt;?php if (\$X &lt; 10) {     echo 'X es menor que 10'; } else {     echo 'x es mayor que 10'; } ?&gt;</pre>	<pre>&lt;?php echo (\$X &lt; 10) ? 'X es menor que 10' :     'X es mayor que 10'; ?&gt;</pre>

Aquí, el operador ternario selecciona el código a la izquierda de los dos puntos (`:`) si la evaluación de la condición es verdadera, y el código a la derecha de los dos puntos en caso de que la condición sea falsa.

## Prueba esto 3-1 Probar números pares y nones

Ahora que ya conoces las bases de las declaraciones condicionales, veamos un ejemplo de cómo pueden utilizarse. El siguiente programa solicitará al usuario que ingrese un número en un formulario Web, revisará si es un par o non y regresará el mensaje correspondiente.

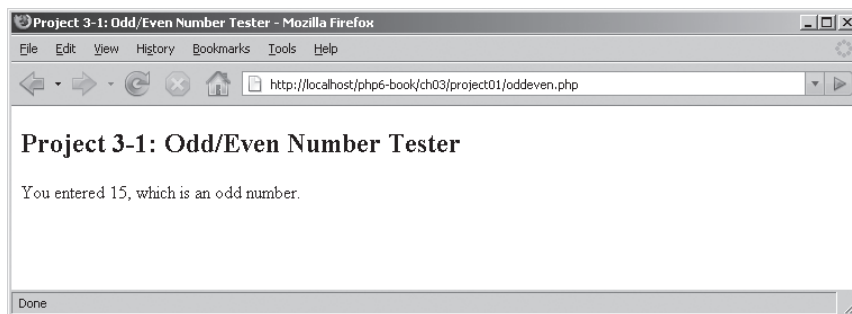
He aquí el código (*paresnones.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 3-1: Verificador de Números Pares y Nones</title>
  </head>
  <body>
    <h2>Proyecto 3-1: Verificador de Números Pares y Nones</h2>
  <?php
    // si el formulario aún no ha sido enviado
    // muestra el formulario
    if (!isset ($_POST['submit'])) {
  ?>
    <form method="post" action="paresnones.php">
      Ingrese un número: <br />
      <input type="text" name="num" size="3" />
      <p>
        <input type="submit" name="submit" value="Enviar" />
      </p>
    </form>
  <?php
    // si el formulario ha sido enviado
    // procesa los datos de entrada del formulario
    } else {
      //recupera el número enviado en el formulario
      $num = $_POST['num'];

      // prueba el valor para los números nones
      // muestra el mensaje apropiado
      if (($num % 2) == 0) {
        echo 'Usted ha escrito ' . $num . ', que es un número par.';
      } else {
        echo 'Usted ha escrito ' . $num . ', que es un número non.';
      }
    }
  ?>
  </body>
</html>
```

(continúa)





**Figura 3-1** Probar números pares y nones con PHP

Este programa consta de dos secciones: la primera mitad genera un formulario Web para que el usuario inserte un número y la segunda averigua si el número es par o non y presenta el mensaje correspondiente. En casi todos los casos, estas dos secciones estarían en archivos separados, pero han sido combinadas en un solo script PHP gracias a la magia de las declaraciones condicionales.

¿Cómo funciona esto? Bien, cuando se envía el formulario Web, la variable `$_POST` contendrá una entrada para el elemento `<input type='submit' . . . >`. Después, una prueba condicional comprueba la presencia o ausencia de esta variable: en caso de estar ausente, el programa “sabe” que el formulario Web no ha sido enviado todavía y ejecuta el código HTML del formulario; en caso de estar presente, el programa “sabe” que el formulario ha sido enviado y procede a probar el valor enviado.

La prueba para verificar si el valor enviado es un número non es realizada por una segunda declaración condicional `if-else`. Aquí, la expresión condicional consiste en dividir el valor de entrada entre 2 y comprobar que el residuo sea igual a cero. Si la prueba regresa un valor verdadero, se presenta en pantalla el mensaje de los números pares; de lo contrario, se presenta el mensaje correspondiente a los números nones.

La figura 3-1 muestra un ejemplo de los datos de salida.

## Escribir declaraciones condicionales más complejas

La declaración `if-else` te permite definir acciones para dos posibilidades: una condición verdadera y una falsa. Sin embargo, es posible que en realidad tu programa tenga que decidir entre más de estos dos sencillos resultados. Para tales situaciones, PHP ofrece dos constructores que permiten al programador manejar múltiples posibilidades: la declaración `if-elseif-else` y la declaración `switch-case`.

## La declaración if-elseif-else

La declaración `if-elseif-else` te permite unir varias declaraciones tipo `if-else`, permitiendo que el programador pueda definir acciones para más de dos resultados posibles. Examina el siguiente ejemplo, que ilustra su uso:

```
<?php
// maneja varias posibilidades
// define un mensaje diferente para cada día
$hoy = 'Martes';
if ($hoy == 'Lunes'){
    echo 'El lunes la cara del niño está limpia.';
} elseif ($hoy == 'Martes'){
    echo 'El martes el niño está lleno de gracia.';
} elseif ($hoy == 'Miércoles'){
    echo 'El miércoles el niño está lleno de preocupaciones.';
} elseif ($hoy == 'Jueves'){
    echo 'El jueves el niño se tiene que ir.';
} elseif ($hoy == 'Viernes'){
    echo 'El viernes el niño es amoroso y dadivoso.';
} elseif ($hoy == 'Sábado'){
    echo 'El sábado el niño trabaja duro.';
} else {
    echo 'No hay información disponible para este día';
}
?>
```

En este caso, el programa mostrará diferentes mensajes para cada día de la semana (dependiendo de lo que se establezca en la variable `$hoy`). Advierte también la última rama `else`: se trata de una rama para “captar el resto”, que será accionada en caso de que ninguna de las declaraciones condicionales anteriores resulten verdaderas; es una manera muy útil para cubrir situaciones imprevisibles.

Hay algo muy importante para recordar sobre el constructor `if-elseif-else`: en cuanto la primera declaración condicional resulte verdadera, PHP ejecutará el código correspondiente, se saltará el resto de las pruebas condicionales e irá directo a las líneas posteriores del bloque entero `if-elseif-else`. Así pues, aunque más de una declaración sea verdadera, PHP sólo ejecutará el código correspondiente a la primera que sea evaluada de manera positiva.

## La declaración switch-case

Una opción a `if-elseif-else` es `switch-case`, que realiza casi la misma acción: co-  
teja una variable contra una serie de valores hasta que localiza una coincidencia, para luego

ejecutar el código correspondiente a la misma. Examina el siguiente código, equivalente al ejemplo inmediato anterior:

```
<?php
// maneja varias posibilidades
// define un mensaje diferente para cada día
$hoY = 'Martes';
switch ($hoY){
    case 'Lunes':
        echo 'El lunes la cara del niño está limpia.';
        break;
    case 'Martes':
        echo 'El martes el niño está lleno de gracia.';
        break;
    case 'Miércoles':
        echo 'El miércoles el niño está lleno de preocupaciones.';
        break;
    case 'Jueves':
        echo 'El jueves el niño se tiene que ir.';
        break;
    case 'Viernes':
        echo 'El viernes el niño es amoroso y dadivoso.';
        break;
    case 'Sábado':
        echo 'El sábado el niño trabaja duro.';
        break;
    default:
        echo 'No hay información disponible para este día';
        break;
}
?>
```

El constructor `switch-case` presenta una importante diferencia en comparación con el constructor `if-elseif-else`: una vez que PHP encuentra una declaración `case` que es evaluada como verdadera, ejecuta no sólo el código correspondiente, sino todo el código de las demás declaraciones `case`. Si esto no es lo que quieres, debes añadir la declaración `break` al final de cada bloque `case` (como se hizo en el ejemplo anterior), para indicar a PHP que termine la ejecución del constructor una vez que haya ejecutado el código correspondiente al primer caso evaluado como verdadero.

Advierte también el caso `'default'`: como su nombre lo sugiere, especifica el conjunto de acciones por defecto que PHP debe tomar cuando ninguno de los anteriores casos se evalúe como verdadero. Este caso por defecto, al igual que la rama `else` del bloque `if-elseif-else`, es útil para “captar el resto” en situaciones imprevistas.

**Prueba esto 3-2****Asignar niños exploradores  
a su tienda de campaña**

Ahora utilizaremos la declaración `if-elseif-else` para crear una pequeña aplicación para los líderes de niños exploradores de cualquier parte: una herramienta Web que asigna automáticamente la tienda de campaña correcta a cada niño explorador durante el campamento, basándose en su edad. La aplicación muestra a los exploradores un formulario Web en el cual pueden escribir su edad y luego les asigna una de cuatro tiendas de campaña: Roja, Verde, Azul y Negra, que compartirán con otros exploradores de casi la misma edad.

He aquí el código (*tienda.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 3-2: Asignación de tiendas de campaña</title>
  </head>
  <body>
    <h2>Proyecto 3-2: Asignación de tiendas de campaña</h2>
  <?php
    // si el formulario no ha sido enviado
    // muestra el formulario
    if (!isset($_POST['submit'])) {
  ?>
    <form method="post" action="tiendas.php">
      Escribe tu edad: <br />
      <input type="text" name="age" size="3" />
      <p>
        <input type="submit" name="submit" value="Enviar" />
      </form>
  <?php
    // si el formulario ha sido enviado
    // procesa los datos ingresados
    } else {
      // recupera la edad enviada por el método POST
      $edad = $_POST['age'];

      // asigna a una de cuatro tiendas
      // de acuerdo con el "binario" de edad al que pertenece
      if ($edad <= 9){
        echo "Estás en la tienda de campaña Roja.";
      } elseif ($edad > 9 && $edad <= 11) {
        echo "Estás en la tienda de campaña Azul.";
      } elseif ($edad > 11 && $edad <= 14) {
        echo "Estás en la tienda de campaña Verde.";
```

(continúa)



**Figura 3-2** La página resultante, que asigna al usuario una tienda de campaña

```

    } elseif ($edad > 14 && $edad <= 17) {
        echo "Estás en la tienda de campaña Negra.";
    } else {
        echo "Mejor comunícate con el líder de los niños exploradores.";
    }
}
?>
</body>
</html>

```

Como el proyecto anterior de este mismo capítulo, éste también combina el formulario Web y su página resultante en un solo script, separado por la declaración condicional `if-elseif-else`. Una vez que el explorador ingresa su edad en el formulario Web y lo envía, un bloque `if-elseif-else` se encarga de definir cuatro rangos de edad, prueba la edad enviada contra estos rangos y decide cuál tienda de campaña es más apropiada para el explorador. Los rangos de edad son 0-9 (tienda de campaña Roja); 10-11 (tienda Azul); 12-14 (Verde); 15-17 (Negra). Los exploradores mayores de 17 años reciben un mensaje que les indica comunicarse con el líder para arreglar su acomodo.

La figura 3-2 muestra la página resultante.

## Combinar declaraciones condicionales

PHP permite que una declaración condicional se anide dentro de otra, con el fin de tomar decisiones más complejas. Para ilustrarlo, examina el siguiente código:

```

<?php
// los empleados con ingresos mensuales <= 15000
// y con un rendimiento >= 3 reciben un bono de $5000
www.detodoprogramacion.com

```

```
// el resto recibe un bono de $3000
if ($rendimiento >= 3) {
    if ($salario < 15000) {
        $bono = 5000;
    }
} else {
    if ($salario > 15000){
        $bono = 3000;
    }
}
?>
```

También puedes combinar declaraciones condicionales utilizando operadores lógicos, como `&&` o `||`. Los estudiaste en el capítulo anterior; la tabla 3-2 hace una recapitulación rápida de ellos.

He aquí un ejemplo de la manera en que estos operadores pueden utilizarse con una declaración condicional:

```
<?php
$fecha = 2008;
// los años bisiestos son divisibles entre 400
// o entre 4, pero no entre 100
if (($fecha % 400 == 0) || (($fecha % 100 != 0) && ($fecha % 4 == 0))) {
    echo "$fecha es año bisiesto.";
} else {
    echo "$fecha no es año bisiesto.";
}
?>
```

## Repetir acciones con bucles

Como cualquier buen lenguaje de programación, PHP da soporte a *bucles*: en esencia, con este nombre se designa la capacidad de repetir una serie de acciones hasta que se cumple una condición especificada. Los bucles son una herramienta importante que ayuda a volver automáticas tareas repetitivas dentro del programa. PHP da soporte a cuatro tipos diferentes

Operador	Descripción
<code>&amp;&amp;</code>	Y (AND)
<code>  </code>	O (OR)
<code>!</code>	NO (NOT)

**Tabla 3-2** Operadores lógicos comunes

de bucles, tres de los cuales son presentados en la siguiente sección (el cuarto se explica en el siguiente capítulo).

## El bucle while

El bucle más sencillo de entender es el `while`, que se repite continuamente mientras una condición específica sea verdadera. A continuación presentamos un ejemplo, que utiliza un bucle para escribir una 'x' de manera repetida en la página de salida.

```
<?php
// repite constantemente hasta que el contador llega a 10
// datos de salida: 'xxxxxxxxxx'
$contador = 1;
while ($contador < 10) {
    echo 'x';
    $contador++;
}
?>
```

Observa la condición encerrada entre paréntesis: mientras sea verdadera, se ejecutará el código entre las llaves. En cuanto la condición sea falsa, el bucle se detiene y las líneas que le siguen se ejecutan de la manera usual.

## El bucle do-while

Con el bucle `while`, la condición que habrá de evaluarse se prueba al principio de cada repetición del bucle. Existe una variante, el bucle `do-while`, que evalúa la condición al final de cada repetición. He aquí una revisión del ejemplo anterior para ilustrar lo dicho:

```
<?php
// repite constantemente hasta que el contador llega a 10
// datos de salida: 'xxxxxxxxxx'
$contador = 1;
do {
    echo 'x';
    $contador++;
} while ($contador < 10);
?>
```

La diferencia en la estructura también debe ser evidente: con un bucle `do-while`, la condición que habrá de evaluarse ahora aparece en la parte inferior del bloque de código, en lugar de hacerlo al principio.

**NOTA**

Las diferencias en el comportamiento entre un bucle `while` y uno `do-while` tienen una importante implicación: en un bucle `while`, si la expresión condicional es evaluada como falsa en la primera pasada, el bucle nunca se ejecutará. En cambio, el bucle `do-while` siempre se ejecutará por lo menos una vez, aunque la expresión condicional sea falsa, porque la condición se evalúa al final de la repetición del bucle y no al principio.

## El bucle `for`

Los bucles `while` y `do-while` son muy sencillos: se repiten mientras la condición especificada sea verdadera. Pero PHP también da soporte a un tipo de bucle más complejo: `for`, que es útil cuando necesitas ejecutar un conjunto de declaraciones un número determinado de veces.

La mejor manera de comprender el bucle `for` es analizando el código donde aparece. He aquí un ejemplo, que lista los números del 1 al 10:

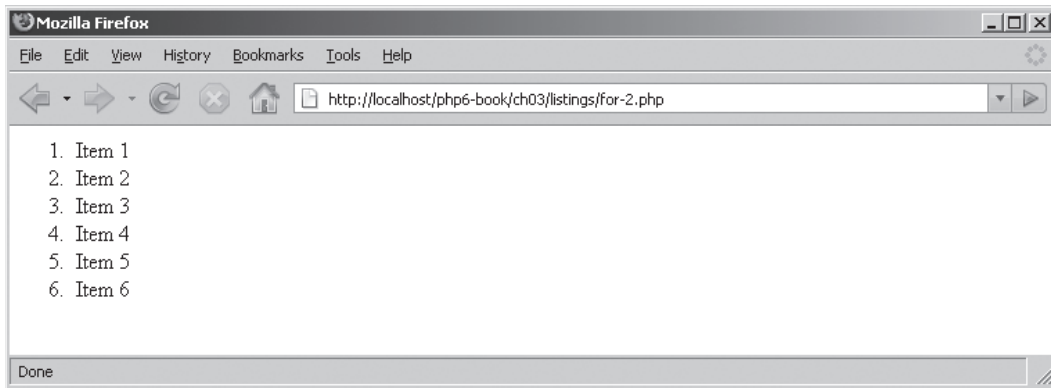
```
<?php
// repite constantemente hasta que el contador llega a 10
// datos de salida:
for ($x=1; $x<10; $x++) {
    echo "$x ";
}
?>
```

En este listado, el bucle comienza asignando a la variable contador `$x` el valor 1; después ejecuta la declaración que conforma el bucle. Una vez que alcanza el final de la primera repetición, actualiza el contador añadiéndole 1, revisa la expresión condicional para asegurarse de que aún no ha alcanzado el valor predeterminado (10), y ejecuta el bucle una vez más. El proceso continúa hasta que el contador alcanza el valor de 10 y la expresión condicional se vuelve falsa.

Como se aprecia en el ejemplo anterior, un bucle `for` típico requiere el uso de tres expresiones, encerradas entre paréntesis y separadas por puntos y comas:

- La primera es una expresión de asignación, que inicializa el bucle con cierto valor; en este caso le asigna el valor 1 a la variable `$x`.
- La segunda es una expresión condicional, que debe evaluarse como verdadera o falsa; el bucle seguirá ejecutándose mientras esta condición sea verdadera. Una vez que se vuelva falsa, el bucle dejará de ejecutarse.
- La tercera es, de nuevo, una expresión de asignación, que se ejecuta al final de cada repetición del bucle, y que actualiza el contador con un nuevo valor; en este caso añade 1 al valor de `$x`.





**Figura 3-3** Una lista generada dinámicamente

He aquí otro ejemplo; éste presenta el uso del bucle `for` para generar una lista ordenada en HTML:

```
<?php
// genera una lista ordenada con 6 elementos
echo "<ol>";
for ($x=1; $x<7; $x++) {
    echo "<li>Elemento $x</li>";
}
echo "</ol>";
?>
```

La figura 3-3 muestra cómo se presentan los datos de salida.

## Combinar bucles

Al igual que con las declaraciones condicionales, también es posible anidar un bucle dentro de otro. Para ilustrar esto, examina el siguiente ejemplo, que anida un bucle `for` dentro de otro para generar dinámicamente una tabla HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml11-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title></title>
  </head>
  <body>
    <?php
    // genera una tabla HTML
    // 3 filas, 4 columnas
    echo "<table border=\"1\">";
```

```

for ($fila=1; $fila<4; $fila++) {
    echo "<tr>";
    for ($col=1; $col<5; $col++) {
        echo "<td>Fila $fila, Columna $col</td>";
    }
    echo "</tr>";
}
echo "</table>";
?>
</body>
</html>

```

Este script utiliza dos bucles `for` anidados. El bucle externo es el responsable de generar las filas de la tabla y se ejecuta tres veces. En cada repetición de este bucle externo, también se dispara el bucle interno, que es responsable de generar las celdas dentro de cada fila (equivalente a las columnas) y se ejecuta cuatro veces. El resultado final es una tabla con tres filas, cada una de ellas con cuatro celdas. La figura 3-4 muestra el resultado.

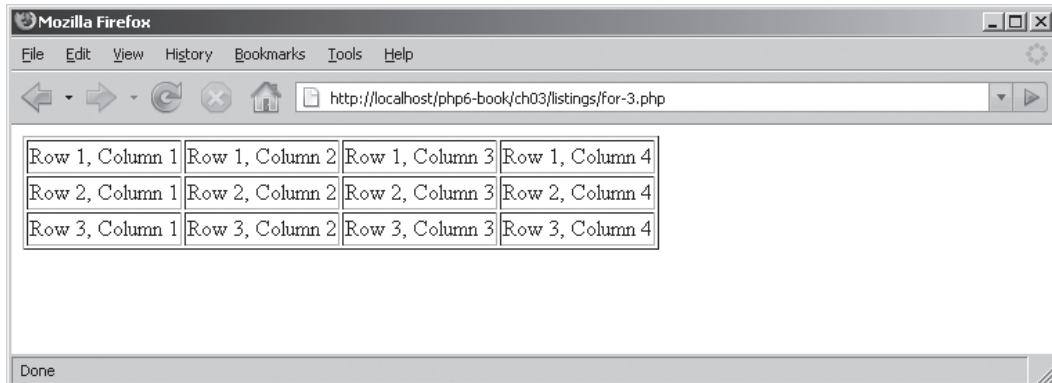
## Interrumpir y omitir bucles

Ya que estamos en el tema de los bucles, es interesante tratar dos declaraciones que te permiten interrumpir u omitir una iteración particular de un bucle. La declaración PHP `break` permite romper un bucle en cualquier punto. Para ilustrarlo, examina el siguiente bucle que normalmente se repetiría cinco veces, pero que se detiene después de la segunda iteración debido a la declaración `break`:

```

<?php
$cuenta = 0;
// recorre el bucle 5 veces

```



**Figura 3-4** Tabla generada dinámicamente

```
while ($cuenta <= 4) {  
    $cuenta++;  
    // cuando el contador llega a 3  
    // rompe el bucle  
    if ($cuenta == 3) {  
        break;  
    }  
    echo "Esta es la iteración #$cuenta <br/>";  
}  
?>
```

A diferencia de `break`, `continue` no detiene el procesamiento del bucle; simplemente “salta una” iteración. Para ver cómo trabaja esto, considere el siguiente bucle, que se itera cinco veces pero omite la tercera iteración debido a la intervención de `continue`:

```
<?php  
$cuenta = 0;  
// recorre el bucle 5 veces  
while ($cuenta <= 4) {  
    $cuenta++;  
    // cuando el contador llega a 3  
    // omite la siguiente iteración  
    if ($cuenta == 3) {  
        continue;  
    }  
    echo "Esta es la iteración #$cuenta <br/>";  
}  
?>
```

## Prueba esto 3-3 Construir una calculadora factorial

Una aplicación sencilla y real que puedes probar por ti mismo para comprender mejor la manera en que funcionan los bucles, es una calculadora factorial. En caso de que te hayas dormido en la clase de matemáticas el día que se explicó, el factorial de un número  $n$  es simplemente el producto de la multiplicación de todos los números entre  $n$  y 1. Así, por ejemplo, el factorial de 4 es  $4 * 3 * 2 * 1 = 24$ .

La calculadora de factoriales que construirás en esta sección es interactiva: solicita que el usuario escriba un número en un formulario Web y luego calcula el factorial de ese número. He aquí el código (*factorial.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
    "DTD/xhtml11-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">  
    <head>  
        <title>Proyecto 3-3: Calculadora Factorial</title>  
        www.detodoprogramacion.com
```

```

</head>
<body>
  <h2>Proyecto 3-3: Calculadora Factorial</h2>
<?php
  // si el formulario no se ha enviado
  // presenta el formulario
  if (!isset ($_POST['submit'])) {
?>
    <form method="post" action="factorial.php">
      Ingresa un número: <br />
      <input type="text" name="num" size="3" />
      <p>
        <input type="submit" name="submit" value="Enviar" />
      </form>
<?php
  // si el formulario ya se ha enviado
  // procesa los datos de entrada
  } else {
    // recupera el número del formulario
    $num = $_POST['num'];

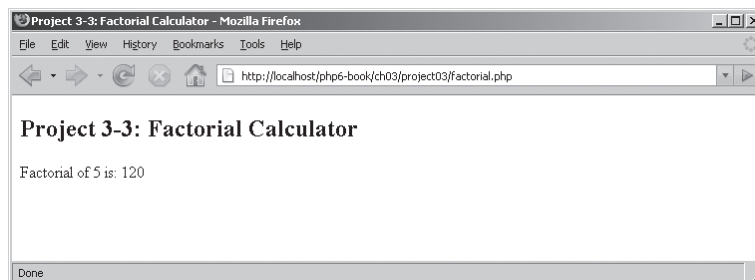
    // verificar que el número sea positivo
    if ($num <= 0) {
      echo 'Error: por favor ingrese un número superior a 0';
      exit();
    }

    // calcula el factorial
    // multiplicando el número por
    // todos los números entre él y el 1
    $factorial = 1;
    for ($x=$num; $x>=1; $x--){
      $factorial *= $x;
    }
    echo "El factorial de $num es: $factorial";
  }
?>
</body>
</html>

```

La mayor parte del trabajo es realizada por el bucle `for`. Este bucle contador se inicia con el número ingresado por el usuario en el formulario Web y luego comienza a contar hacia atrás, disminuyendo el contador a razón de 1 en cada iteración. Cada vez que el bucle se ejecuta, el producto previo calculado se multiplica por el valor actual que contiene el contador. El resultado final es el factorial del número ingresado por el usuario.

La figura 3-5 muestra el resultado después de enviar el formulario.



**Figura 3-5** La página de salida, que muestra el factorial del número

## Trabajar con funciones de cadenas de texto y numéricas

En el capítulo anterior aprendiste un poco sobre los tipos de datos de PHP, incluidos sus operadores para manipular cadenas de texto y números. Pero PHP te permite hacer mucho más que una simple unión de caracteres y sumar valores; el lenguaje incluye una biblioteca completa de funciones integradas que te permite hacer casi cualquier cosa, desde dividir cadenas de texto e invertir el orden de las letras hasta calcular valores logarítmicos. Este capítulo presenta una introducción a algunas de esas funciones.

### Utilizar funciones de cadena de texto

PHP tiene más de 75 funciones integradas para la manipulación de texto, que dan soporte a operaciones que van desde repetición e inversión de cadenas de texto hasta comparaciones y operaciones de búsqueda y sustitución. La tabla 3-3 presenta una lista con algunas de estas funciones.

#### Buscar cadenas de texto vacías

La función `empty()` regresa un valor verdadero si una variable de cadena de texto está “vacía”. Las variables de cadena de texto vacías son las que tienen valores `' '`, `0`, `'0'` o `NULL`. La función `empty()` también regresa un valor verdadero cuando se utiliza con una variable inexistente. A continuación unos ejemplos:

```
<?php
// prueba si la cadena de caracteres está vacía
// datos de salida: true
$cadena = '';
echo (boolean) empty($cadena);

// datos de salida: true
$cadena = null;
```

```
echo (boolean) empty($cadena);
// datos de salida: true
$cadena = '0';
echo (boolean) empty($cadena);

// datos de salida: true
unset($cadena);
echo (boolean) empty($cadena);
?>
```

Función	Lo que hace
empty()	Prueba si una cadena de caracteres está vacía
strlen()	Calcula la cantidad de caracteres en una cadena
strrev()	Invierte una cadena de caracteres
str_repeat()	Repite una cadena de caracteres
substr()	Recupera una sección de la cadena de caracteres
strcmp()	Compara dos cadenas de caracteres
str_word_count()	Calcula la cantidad de palabras en una cadena de caracteres
str_replace()	Reemplaza partes de la cadena de caracteres
trim()	Elimina espacios vacíos al principio y al final de una cadena de caracteres
strtolower()	Convierte a minúsculas los caracteres de una cadena
strtoupper()	Convierte a mayúsculas los caracteres de una cadena
ucfirst()	Convierte a mayúsculas el primer carácter de una cadena
ucwords()	Convierte a mayúsculas el primer carácter de cada palabra en una cadena
addslashes()	Crea caracteres de escape especiales en una cadena con diagonales invertidas
stripslashes()	Elimina diagonales invertidas de una cadena de caracteres
htmlentities()	Codifica en HTML dentro de una cadena de caracteres
htmlspecialchars()	Codifica caracteres especiales HTML dentro de una cadena
nl2br()	Reemplaza saltos de línea con elementos  
html_entity_decode()	Decodifica entidades HTML dentro de una cadena de caracteres
htmlspecialchars_decode()	Decodifica caracteres especiales HTML dentro de una cadena
strip_tags()	Elimina el código PHP y HTML de una cadena de caracteres

**Tabla 3-3** Funciones comunes de cadena de caracteres en PHP

## Invertir y repetir cadenas de caracteres

La función `strlen()` regresa la cantidad de caracteres en una cadena. He aquí un ejemplo de ésta en acción:

```
<?php
// calcula la longitud de una cadena de caracteres
// datos de salida: 17
$cadena = 'Bienvenido a Xanadu';
echo strlen($cadena);
?>
```

Invertir una cadena de caracteres es tan sencillo como invocar la función `strrev()`, como en el siguiente ejemplo:

```
<?php
// invertir cadena de caracteres
// datos de salida: 'osap oñeuqep nU'
$cadena = 'Un pequeño paso';
echo strrev($cadena);
?>
```

En caso de que necesites repetir una cadena de caracteres, PHP ofrece la función `str_repeat()`, que acepta dos argumentos: la cadena de caracteres y la cantidad de veces que será repetida. He aquí un ejemplo:

```
<?php
// repetir cadena de caracteres
// datos de salida: 'yoyoyo'
$cadena = 'yo';
echo str_repeat($cadena, 3);
?>
```

## Trabajar con subcadenas de caracteres

PHP también te permite dividir una cadena de caracteres en partes más pequeñas con la función `substr()`, que acepta tres argumentos: la cadena de caracteres original, la posición (*offset*) donde debe comenzar la división y el número de caracteres que debe mostrar a partir de la posición donde comienza. El siguiente ejemplo lo ilustra:

```
<?php
// extrae subcadena
// datos de salida: 'venido'
$cadena = 'Bienvenido a ninguna parte';
echo substr($cadena, 3, 4);
?>
```

## NOTA

Cuando utilices la función `substr()`, el primer carácter de la cadena se considera como `offset0`, el segundo como `offset1`, etcétera.

Para extraer una subcadena desde el final de la cadena original (en lugar de hacerlo desde el principio), pasa un valor negativo de `offset` en `substr()`, como se muestra en la siguiente revisión del ejemplo anterior:

```
<?php
// extrae una subcadena
// datos de salida: 'un arte'
$cadena = 'Bienvenido a ninguna parte';
echo substr($cadena, 3, 5) . substr($cadena, -4, 4);
?>
```

## Pregunta al experto

**P:** ¿Puedo recuperar un solo carácter de una cadena? ¿Cómo?

**R:** Existen dos maneras de recuperar un solo carácter de una cadena. El camino largo utiliza la función `substr()`, como se muestra en el siguiente ejemplo:

```
<?php
// datos de salida: 'r'
$cadena = 'abracadabra';
echo substr($cadena, 2, 1);
?>
```

La función `substr()` acepta tres argumentos: la cadena de caracteres original, la posición donde debe comenzar la extracción y el número de caracteres a extraer comenzando por la posición especificada. Ten en mente que la cuenta de la posición comienza en 0 y no en 1.

De cualquier manera, PHP también da soporte a una abreviatura de sintaxis para completar la misma tarea. Consiste en encerrar entre llaves la posición específica del carácter que quieres recuperar; las llaves se colocan justo después del nombre de la variable. El siguiente ejemplo lo ilustra:

```
<?php
// datos de salida: 'r'
$cadena = 'abracadabra';
echo $cadena{2};
?>
```



## Comparar, contar y reemplazar cadenas de caracteres

Si necesitas comparar dos cadenas de caracteres, la función `strcmp()` realiza una comparación sensible a mayúsculas, regresa un valor negativo en caso de que la primera sea “menor” que la segunda; en caso contrario regresa un valor positivo y regresa un cero si ambas cadenas son “iguales”. He aquí unos ejemplos de cómo funciona:

```
<?php
// compara cadenas de caracteres
$a = 'hola';
$b = 'hola';
$c = 'hola';

// datos de salida: 0
echo strcmp($a, $b);

// datos de salida: 1
echo strcmp($a, $c);
?>
```

La función PHP `str_word_count()` proporciona una manera fácil de contar el número de palabras en una cadena de caracteres. El siguiente ejemplo ilustra su uso:

```
<?php
// cuenta palabras
// datos de salida: 6
$cadena = "Mi nombre es Bond, James Bond";
echo str_word_count($cadena);
?>
```

Si necesitas sustituir caracteres dentro de una cadena, PHP también cuenta con la función `str_replace()`, diseñada especialmente para realizar operaciones de búsqueda y sustitución. Esta función acepta tres argumentos: el término que se busca, el término que lo reemplaza y la cadena de caracteres donde se debe realizar la sustitución. He aquí un ejemplo:

```
<?php
// reemplaza '@' por 'arroba'
// datos de salida: 'juan arroba dominio.net'
$cadena = 'juan@dominio.net';
echo str_replace('@', ' arroba ', $cadena);
?>
```

## Dar formato a cadenas de caracteres

La función PHP `trim()` puede utilizarse para eliminar espacios en blanco al principio o al final de una cadena de caracteres; esto es muy útil cuando se procesan datos de entrada provenientes de un formulario Web. He aquí un ejemplo:

```
<?php
// eliminar espacios en blanco al principio y al final
// datos de salida: 'a b c'
$cadena = ' a b c ';
echo trim($cadena);
?>
```

Cambiar a mayúsculas o minúsculas los caracteres de una cadena es tan sencillo como invocar las funciones `strtolower()` para minúsculas y `strtoupper()` para mayúsculas, como se muestra en los siguientes ejemplos:

```
<?php
// cambia las mayúsculas por minúsculas
$cadena = 'Yabba Dabba Doo';

// datos de salida: 'yabba dabba doo'
echo strtolower($cadena);

// datos de salida: 'YABBA DABBA DOO'
echo strtoupper($cadena);
?>
```

También puedes cambiar a mayúsculas el primer carácter de una cadena con la función `ucfirst()`, o bien el primer carácter de cada palabra de la cadena con la función `ucwords()`. El siguiente ejemplo muestra ambas funciones:

```
<?php
// cambia mayúsculas/minúsculas
$cadena = 'las brigadas amarillas';

// datos de salida: 'Las Brigadas Amarillas'
echo ucwords($cadena);

// datos de salida: 'Las brigadas amarillas'
echo ucfirst($cadena);
?>
```

## Trabajar con cadenas de caracteres HTML

PHP cuenta con funciones exclusivas para trabajar con cadenas de caracteres HTML. Antes que nada, está la función `addslashes()`, la cual busca signos especiales y los convierte automáticamente en caracteres de escape, añadiéndoles una diagonal invertida. He aquí un ejemplo:

```
<?php
// convierte signos especiales en caracteres de escape
// datos de salida: las décadas de los 80\'s y 90\'s
$cadena = "las décadas de los 80's y 90's";
echo addslashes($cadena);
?>
```

Puedes revertir el trabajo hecho por `addslashes()` con la función `stripslashes()`, que elimina todas las diagonales invertidas de una cadena de caracteres. Examina el siguiente ejemplo que ilustra lo escrito:

```
<?php
// elimina las diagonales invertidas
// datos de salida: 'Juan D'Souza dijo "Nos vemos luego".'
$cadena = "Juan D\'Souza dijo \"Nos vemos luego\".";
echo stripslashes($cadena);
?>
```

Las funciones `htmlentities()` y `htmlspecialchars()` convierten automáticamente los símbolos especiales HTML (como `<` y `>`) en su correspondiente código (`&lt;` y `&gt;`). De manera similar, la función `nl2br()` reemplaza automáticamente los caracteres correspondientes a saltos de línea en una cadena con la etiqueta HTML correspondiente: `<br />`. He aquí un ejemplo:

```
<?php
// reemplazar con entidades HTML
// datos de salida: '&lt;div width=&quot;200&quot;&gt;
//                      Esta es una etiqueta div&lt;/div&gt;
$html = '<div width="200">Esta es una etiqueta div</div>';
echo htmlentities($html);

// reemplaza saltos de línea con elementos <br />
// datos de salida: 'Esta es una línea ro<br>
//                      ta'
$lineas = 'Esta es una línea ro
          ta';
echo nl2br($lineas);
?>
```

Puedes revertir los efectos de las funciones `htmlentities()` y `htmlspecialchars()` con las funciones `html_entity_decode()` y `htmlspecialchars_decode()`, respectivamente.

Por último, la función `strip_tags()` busca todo el código HTML y PHP dentro de una cadena de caracteres y lo elimina para generar una cadena “limpia”. He aquí un ejemplo:

```
<?php
// elimina las etiquetas HTML de una cadena de caracteres
// datos de salida: 'Por favor inicie sesión de nuevo'
$html = '<div width="200">Por favor <strong>inicie sesión de nuevo</strong></div>';
echo strip_tags($html);
?>
```

## Utilizar funciones numéricas

No pienses que el poder de PHP se limita a las cadenas de texto: el lenguaje cuenta con más de 50 funciones integradas para trabajar con números, desde sencillas funciones de formato hasta funciones aritméticas, logarítmicas y manipulaciones trigonométricas. La tabla 3-4 lista algunas de estas funciones.

### Hacer cálculos

Una tarea común cuando se trabaja con números es redondearlos hacia arriba o hacia abajo. PHP ofrece las funciones `ceil()` y `floor()` para realizar estas tareas, respectivamente, y se muestran en el siguiente ejemplo:

```
<?php
// redondea números
// datos de salida: 19
$num = 19.7;
echo floor($num);
```

Función	Lo que hace
<code>ceil()</code>	Redondea un número hacia arriba
<code>floor()</code>	Redondea un número hacia abajo
<code>abs()</code>	Encuentra el valor absoluto de un número
<code>pow()</code>	Eleva un número a la potencia de otro
<code>log()</code>	Encuentra el logaritmo de un número
<code>exp()</code>	Encuentra el exponente de un número
<code>rand()</code>	Genera un número aleatorio
<code>bindec()</code>	Convierte un número de binario a decimal
<code>decbin()</code>	Convierte un número de decimal a binario
<code>decoct()</code>	Convierte un número de decimal a octal
<code>dechex()</code>	Convierte un número de decimal a hexadecimal
<code>hexdec()</code>	Convierte un número de hexadecimal a decimal
<code>octdec()</code>	Convierte un número de octal a decimal
<code>number_format()</code>	Forma un número con miles y decimales agrupados
<code>printf()</code>	Forma un número utilizando especificaciones personalizadas

**Tabla 3-4** Funciones numéricas PHP comunes

```
// redondea un número hacia arriba
// datos de salida: 20
echo ceil($num);
?>
```

También tenemos la función `abs()`, que regresa el valor absoluto de un número. He aquí un ejemplo:

```
<?php
// regresa el valor absoluto de un número
// datos de salida: 19.7
$num = -19.7;
echo abs($num);
?>
```

La función `pow()` regresa el valor de un número elevado a la potencia de otro:

```
<?php
// calcula 4 ^ 3
// datos de salida: 64
echo pow(4,3);
?>
```

La función `log()` calcula el logaritmo natural o base 10 de un número, mientras que la función `exp()` calcula el exponente de un número. He aquí un ejemplo de ambas en acción:

```
<?php
// calcula el logaritmo natural de 100
// datos de salida: 2.30258509299
echo log(10);

// calcula el logaritmo de 100, base 10
// datos de salida: 2
echo log(100,10);

// calcula el exponente de 2.30258509299
// datos de salida: 9.99999999996
echo exp(2.30258509299);
?>
```

### Generar números aleatorios

Generar números aleatorios con PHP es también una tarea sencilla: la función integrada al lenguaje, `rand()`, genera automáticamente un entero aleatorio mayor que 0. Puedes restringir la generación a cierto rango de números proporcionando límites opcionales como argumentos. El siguiente ejemplo lo ilustra:

```
<?php
// genera un número aleatorio
```

```
echo rand();

// genera un número aleatorio entre 10 y 99
echo rand(10,99);
?>
```

## Convertir entre bases numéricas

PHP cuenta con funciones integradas para hacer conversiones entre diferentes bases numéricas: binario, decimal, octal y hexadecimal. He aquí un ejemplo que muestra en acción las funciones `bindec()`, `decbin()`, `decoct()`, `dechex()`, `hexdec()` y `octdec()`:

```
<?php
// convertir de decimal a binario
// datos de salida: 1000
echo decbin(8);

// convertir de decimal a hexadecimal
// datos de salida: 8
echo dechex(8);

// convertir de decimal a octal
// datos de salida: 10
echo decoct(8);

// convertir de octal a decimal
// datos de salida: 8
echo octdec(10);

// convertir de hexadecimal a decimal
// datos de salida: 101
echo hexdec(65);

// convertir de binario a decimal
// datos de salida: 8
echo bindec(1000);
?>
```

## Formar números

Cuando se trata de formar números, PHP ofrece la función `number_format()`, que acepta cuatro argumentos: el número que se formará, la cantidad de espacios decimales que se deben presentar, el carácter que se utilizará en lugar del punto decimal y el carácter que se empleará para separar los millares agrupados (por lo general se utiliza una coma). Examina el siguiente ejemplo que ilustra lo escrito:

```
<?php
// formar un número (con valores por defecto)
```

```
// datos de salida: 1,106,483
$num = 1106482.5843;
echo number_format($num);

// formar un número (con separadores personalizados)
// datos de salida: 1?106?482*584
echo number_format($num, 3, '*', '?');
?>
```

Para tener mayor control sobre el formato de los números, PHP ofrece las funciones `printf()` y `sprintf()`. Estas funciones, aunque muy útiles, pueden intimidar a los usuarios novatos; por ello, la mejor manera de entenderlas es mediante ejemplos. Examina el siguiente código, que los muestra en acción:

```
<?php
// dar formato de número decimal
// datos de salida: 00065
printf("%05d", 65);

// dar formato de número de punto flotante
// datos de salida: 00239.000
printf("%09.3f", 239);

// dar formato de número octal
// datos de salida: 10
printf("%4o", 8);

// formar un número
// incorporado a una cadena de caracteres
// datos de salida: 'Veó 8 manzanas y 26.00 naranjas'
printf("Veó %4d manzanas y %4.2f naranjas", 8, 26);
?>
```

Ambas funciones aceptan dos argumentos: una serie de *especificadores de formato* y la cadena o el número sin trabajar que va a recibir el formato. Los datos de entrada se forman de acuerdo con las especificaciones anotadas y el resultado se despliega directamente con la función `printf()` o asignándolo a una variable con la función `sprintf()`.

Algunos de los especificadores de formato más comunes se muestran en la tabla 3-5.

También puedes combinar éstos con *especificadores de precisión*, que indican la cantidad de dígitos que habrá de mostrarse para los valores de punto flotante, por ejemplo, `%1.2f` implica que el número debe formarse como valor de punto flotante con dos dígitos desplegados después del punto decimal. En el caso de números más pequeños, es posible añadir también un *especificador de relleno*, que indica a la función que rellene los números hasta cierta longitud específica utilizando un carácter personalizado. Puedes ver estos dos tipos de especificadores en el ejemplo anterior.

Especificador	Lo que significa
%s	Cadena de caracteres
%d	Número decimal
%x	Número hexadecimal
%o	Número octal
%f	Número de punto flotante

**Tabla 3-5** Especificadores de formato para las funciones `printf()` y `sprintf()`

Prueba esto 3-4

Procesar un formulario para registro de miembros

Ahora que ya conoces algunas funciones PHP integradas, utilicemos algo de lo que has aprendido en una aplicación práctica: un formulario de solicitud de membresías para un club deportivo. Este formulario solicitará al aplicante que ingrese información personal; después validará esta información y, de ser aceptable, formulará y enviará un correo electrónico con la información del solicitante al administrador del club.

He aquí el formulario HTML (*registro.html*):

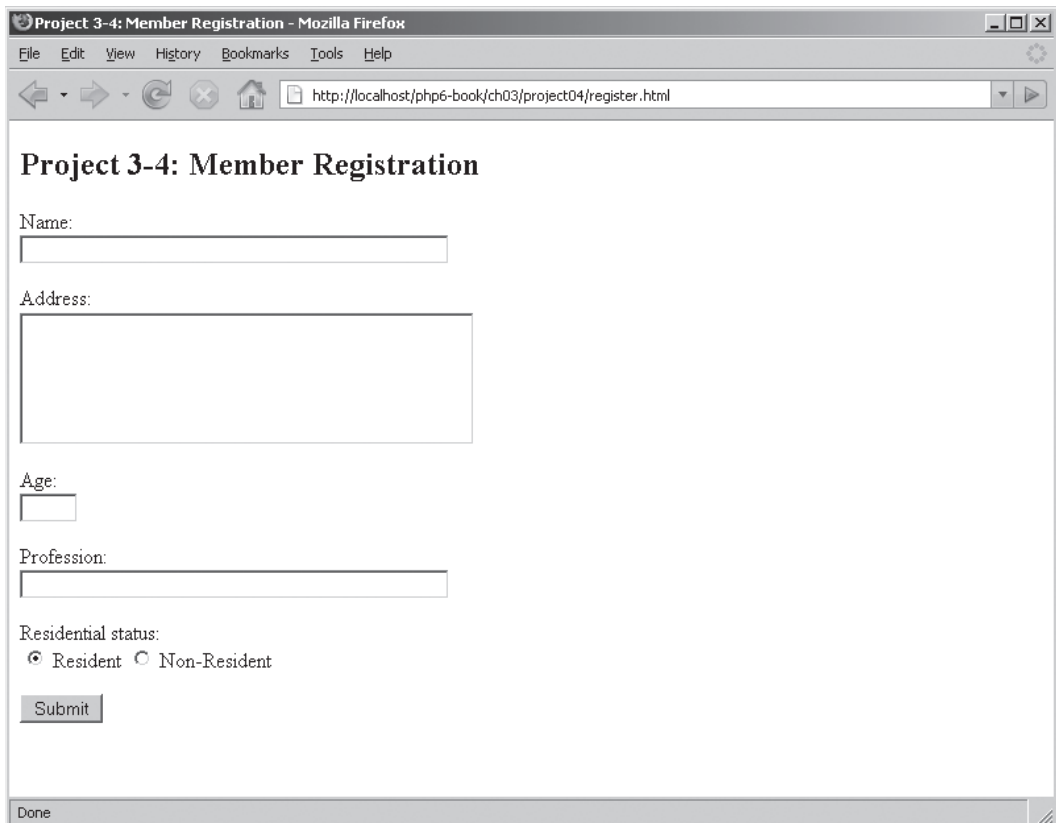
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml11-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 3-4: Registro de Miembros</title>
  </head>
  <body>
    <h2> Proyecto 3-4: Registro de Miembros</h2>
    <form method="POST" action="registro.php">
      Nombre: <br />
      <input type="text" name="nombre" size="50" />
      <p>
        Dirección: <br />
        <textarea name="direccion" rows="5" cols="40"></textarea>
      <p>
        Edad: <br />
        <input type="text" name="edad" size="3" />
      <p>
        Profesión: <br />
        <input type="text" name="profesion" size="50" />
```

(continúa)



```
<p>
Estatus residencial: <br />
<input type="radio" name="residencia" value="yes" checked="true" />
Residente
<input type="radio" name="residencia" value="no" /> No Residente
<p>
<input type="submit" name="submit" value="Enviar" />
</form>
</body>
</html>
```

Este formulario tiene cinco campos de entrada: nombre del solicitante, dirección, edad, profesión y estatus residencial. La figura 3-6 muestra cómo se ve en el explorador Web.



The screenshot shows a Mozilla Firefox browser window with the title "Project 3-4: Member Registration - Mozilla Firefox". The address bar displays "http://localhost/php6-book/ch03/project04/register.html". The form content is as follows:

**Project 3-4: Member Registration**

Name:

Address:

Age:

Profession:

Residential status:  
☒ Resident ☐ Non-Resident

Done

**Figura 3-6** Un formulario de aplicación basado en Web

## NOTA

Con el fin de enviar con éxito un correo electrónico en que PHP utilice la función `mail()`, tu archivo de configuración `php.ini` debe incluir cierta información sobre el servidor de correo electrónico o el agente de correo que va a utilizarse. Los usuarios de Windows necesitarán establecer las opciones `'SMTP'` y `'smtp_port'`, mientras que los usuarios de \*NIX tal vez necesiten establecer la opción `'sendmail_path'`. Para obtener más información sobre estas opciones puedes consultar el manual de PHP en [www.php.net/mail](http://www.php.net/mail).

Una vez que se ha enviado el formulario, los datos ingresados por el solicitante son transmitidos al script de procesamiento (*registro.php*) con el método POST. El siguiente listado muestra el contenido del script:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 3-4: Registro de Miembros</title>
  </head>
  <body>
    <h2> Proyecto 3-4 Registro de Miembros</h2>
  <?php
    // recupera detalles del envío POST
    $nombre = $_POST['nombre'];
    $direccion = $_POST['direccion'];
    $edad = $_POST['edad'];
    $profesion = $_POST['profesion'];
    $residencia = $_POST['residencia'];

    // valida los datos enviados
    // verifica nombre
    if(empty($nombre)){
      die('ERROR: Por favor proporcione su nombre.');
```

```
        die ('ERROR: Por favor proporcione su profesión.');
```

```
    }
    // verificar estatus residencial
    if(strcmp ($residencia, 'no') == 0) {
        die ('ERROR: Las membresías sólo están abiertas para residentes.');
```

```
    }

    // si llegamos a este punto
    // todos los datos de entrada han pasado la validación
    // crea y envía el mensaje de correo electrónico
    $to = 'registro@algun.dominio.com';
    $from = 'webmaster@algun.dominio.com ';
    $subject = 'Solicitud de membresía';
    $body = "Nombre: $nombre\r\nDirección: $direccion\r\n
        Edad: $edad\r\nProfesión: $profesion\r\n
        Estatus residencial: $residencia\r\n";
    if (mail($to, $subject, $body, "From: $from")){
        echo 'Gracias por enviar su solicitud.';
    } else {
        die ('ERROR: Error al enviar el mensaje');
```

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

## NOTA

Recuerda cambiar el valor de la variable `$to` en el script *registro.php*, con tu dirección de correo electrónico.

Este script principia por recuperar los valores enviados por el usuario de `$_POST` y asignando estos valores a variables regulares PHP. Después, se utiliza la función `empty()` para verificar si los campos vienen vacíos; en caso de que cualquier campo no contenga información, se genera un mensaje de error y el script termina de inmediato. Dos pruebas condicionales adicionales están presentes en esta sección: la primera es la edad del solicitante, que filtra solicitantes menores de 18 años y mayores de 60; y la segunda para el estatus residencial del solicitante, que filtra a quienes no son residentes.

Si todo es correcto, el script procede a crear un mensaje de correo electrónico, estableciendo las variables para el remitente, el destinatario, el asunto y el cuerpo del mensaje. Después, estas variables son enviadas a la función PHP `mail()`, que se encarga en realidad de hacer el trabajo pesado para enviar el mensaje de correo electrónico. Si la transmisión de éste tiene éxito, aparece la notificación correspondiente; en caso contrario, se genera un mensaje de error.

La función `mail()` es nueva para ti, por lo que merece una revisión más detallada. La función `mail()` está integrada a PHP para enviar mensajes de correo electrónico y acepta cuatro parámetros: la dirección de correo electrónico del destinatario, el asunto, el cuerpo del mensaje

y una lista adicional de encabezados del mensaje (de los cuales el encabezado 'From' es obligatorio). Utiliza estos parámetros para construir un mensaje de correo electrónico, conectarse a un servidor de correo específico y entregar el mensaje para su envío. Si la entrega tiene éxito, `mail()` regresa un valor verdadero (true); en cualquier otro caso regresa un valor falso (false).

### **PRECAUCIÓN**

Es importante entender el significado del valor que regresa de la función `mail()`. Si regresa un valor verdadero (true), significa que el mensaje se entregó correctamente al servidor de correo para su envío posterior. *No significa* que el mensaje haya sido transmitido al destinatario, o que éste lo haya recibido, porque PHP no tiene medios para rastrear el mensaje una vez que ha sido entregado al servidor de correo. La falta de comprensión de esta diferencia es un error común entre los programadores novatos en PHP.

La función `die()` presentada en este script también es nueva para ti: proporciona una manera muy conveniente de terminar de inmediato la ejecución del script, por lo general cuando ocurre un error. También puedes pasar un mensaje opcional a `die()`; este mensaje será presentado por PHP en el punto de terminación del script, por lo que sirve como explicación útil para que el usuario sepa qué fue lo que salió mal.

## Pregunta al experto

**P:** ¿Puedo añadir encabezados personalizados a un mensaje de correo electrónico enviado por PHP?

**R:** Sí. El cuarto argumento de la función `mail()` es una cadena de caracteres que contiene tu encabezado personalizado, en formato `header:value`. Si tienes más de un encabezado personalizado para añadir, separa los encabezados con `\r\n`. El ejemplo siguiente lo ilustra:

```
<?php
// define mensaje
$to = 'bacchus@vsnl.com';
$subject = 'Hola';
$body = "Esta es una prueba";
// define encabezados personalizados
$headers = "From:webmaster@my.domain.com\r\n
  Organización:MyOrg Inc.\r\nX-Mailer:PHP";
if(mail($to, $subject, $body, $headers)){
    echo 'Su mensaje ha sido enviado.';
} else {
    die('ERROR: Error al enviar el mensaje');
}
?>
```

## Resumen

El objetivo de este capítulo fue que avanzaras en tu curva de aprendizaje, desde la construcción de programas sencillos y lineales hasta la creación de aplicaciones PHP más complejas. Comenzó explicando cómo puedes añadir inteligencia a tus scripts PHP mediante el uso de pruebas condicionales y proporcionó ejemplos del uso de las declaraciones `if`, `if-else`, `if-elseif-else` y `switch-case`. Después fueron presentados los constructores bucle de PHP, mostrándote lo fácil que es automatizar acciones repetitivas con PHP y se presentaron tres tipos comunes de bucles: `while`, `do-while` y `for`.

La segunda mitad del capítulo te llevó por un viaje relámpago a través de las funciones de cadenas de caracteres y numéricas integradas en PHP; se utilizaron ejemplos comentados para explicar la manera de realizar diferentes tareas, desde sencillas comparaciones y extracciones en cadenas de caracteres hasta tareas más complejas de conversión y formato de números. Y como si eso no fuera suficiente, en este capítulo se te guió en el desarrollo de cuatro proyectos, cada uno de ellos diseñado para mostrar un uso práctico de los conceptos aprendidos. Declaraciones condicionales, bucles y funciones integradas... todos ellos fueran aplicados en los proyectos, que van desde una sencilla prueba para identificar números pares y noes hasta una aplicación Web completa que incluyó funciones para la validación de datos de entrada de un formulario y transmisión de un mensaje de correo electrónico.

Al finalizar este capítulo, ahora sabes lo suficiente acerca de la gramática y la sintaxis básica de PHP para comenzar a escribir tus propios scripts PHP con un grado de complejidad razonable. El siguiente capítulo te ayudará a expandir tu arsenal de trucos, al presentarte un nuevo tipo de variable de PHP y al enseñarte a manipular funciones de fecha y hora. Hasta entonces, ocupa un poco de tu tiempo revisando las direcciones Web que se presentan a continuación, que ofrecen información más detallada sobre los temas abordados en este capítulo:

- Declaraciones condicionales y bucles, en [www.php.net/manual/en/language.control-structures.php](http://www.php.net/manual/en/language.control-structures.php)
- Sobre las funciones de cadenas de caracteres de PHP, en [www.php.net/manual/en/ref.strings.php](http://www.php.net/manual/en/ref.strings.php) y [www.melonfire.com/community/columns/trog/article.php?id=88](http://www.melonfire.com/community/columns/trog/article.php?id=88)
- Funciones matemáticas PHP, en [www.php.net/manual/en/ref.math.php](http://www.php.net/manual/en/ref.math.php)
- Envío de mensajes de correo electrónico con PHP, en [www.php.net/manual/en/ref.mail.php](http://www.php.net/manual/en/ref.mail.php)
- Operador ternario de PHP, en [www.php.net/manual/en/language.operators.php](http://www.php.net/manual/en/language.operators.php)



## Autoexamen Capítulo 3

1. ¿Cuál es la diferencia entre una declaración `if-else` y una `if-elseif-else`?
2. Escribe una declaración condicional que verifique el valor de `$ciudad` y que despliegue un mensaje si el valor es igual a `'Minneapolis'`.
3. Explica la diferencia entre los bucles `while` y `do-while`. Ilustra tu respuesta con un ejemplo.
4. Utilizando el bucle `while`, escribe un programa que presente en pantalla la tabla de multiplicar del 8.
5. Rescribe el programa de la pregunta 4 utilizando el bucle `for`.
6. Menciona las funciones que utilizarías para
  - A** Decodificar entidades HTML
  - B** Poner en mayúsculas una cadena de caracteres
  - C** Redondear un número hacia abajo
  - D** Eliminar espacios en blanco de una cadena de caracteres
  - E** Generar un número aleatorio
  - F** Invertir una cadena de caracteres
  - G** Contar las palabras de una cadena de caracteres
  - H** Contar los caracteres de una cadena
  - I** Terminar el procesamiento del script con un mensaje para el usuario
  - J** Comparar dos cadenas de caracteres
  - K** Calcular el exponente de un número
  - L** Convertir un número decimal en un valor hexadecimal
7. ¿Cuáles serían los datos de salida de la siguiente línea de código PHP?:
 

```
<?php printf("%09.6f", 7402.4042); ?>
```
8. Dada la línea `'Marco tuvo un día agradable'`, crea una nueva cadena de caracteres que diga `'Marco tuvo un helado'`.





# Capítulo 4

## Trabajar con matrices



## Habilidades y conceptos clave

- Comprender los diferentes tipos de matrices soportados por PHP
  - Procesar el contenido de matrices con el bucle `foreach`
  - Utilizar matrices con formularios Web
  - Ordenar, fusionar, añadir, modificar y dividir matrices utilizando las funciones integradas de PHP
  - Trabajar con fechas y horas
- 

Por ahora, los tipos simples de datos PHP deben ser pan comido para ti, porque has pasado los últimos capítulos utilizándolos en diferentes permutaciones y combinaciones, para crear aplicaciones de complejidades y usos diversos. Pero hay muchos tipos de datos en PHP, además de cadenas de caracteres, números y booleanos; PHP también da soporte a *matrices*, que te permiten agrupar y manipular más de un valor a la vez.

En este capítulo aprenderás sobre las matrices: cómo crearlas, cómo añadirles elementos y editarlas, cómo manipular los valores que contienen. También conocerás un nuevo tipo de bucle, diseñado exclusivamente para utilizarse con matrices, recibirás un curso rápido sobre algunas funciones integradas de PHP para matrices y aplicarás tus conocimientos en proyectos prácticos.

## Almacenar datos en matrices

Hasta ahora, todas las variables que has utilizado contienen un solo valor. Las variables de matrices son “especiales” porque pueden contener más de un valor a la vez. Eso las hace muy útiles para almacenar valores relacionados (por ejemplo, un conjunto de nombres de frutas), como en el siguiente ejemplo:

```
<?php
// define una matriz
$frutas = array('manzana', 'plátano', 'piña', 'uva');
?>
```

Una vez que tienes una matriz como la anterior, surge una pregunta natural: ¿cómo recuperar un valor específico? Es muy sencillo: se usan números indexados para acceder a los diferentes valores almacenados en la matriz, y el cero representa el primer valor. De esta manera, para acceder al valor 'manzana' de la matriz anterior se utiliza la notación `$frutas[0]`, mientras que el valor 'uva' puede recuperarse con la notación `$frutas[3]`.

PHP también da soporte a un tipo de matriz un poco diferente, en que los números son reemplazados con cadenas de caracteres o “palabras clave” definidas por el usuario. He aquí una versión revisada del ejemplo anterior, que utiliza palabras clave en lugar de números indexados:

```
<?php
// define una matriz
$frutas = array(
    'm' => 'manzana'
    'p' => 'plátano'
    'i' => 'piña'
    'u' => 'uva'
);
?>
```

A este tipo de matriz se le conoce como *matriz asociada*. Las palabras clave de la matriz deben ser únicas; cada una de las palabras clave hace referencia a un solo valor y la relación palabra clave-valor está expresada por el símbolo =>. Para acceder al valor 'manzana' de la matriz, se utiliza la notación `$frutas['m']`, mientras que 'plátano' puede ser recuperado utilizando la notación `$frutas['p']`.

## Asignar valores a matrices

Las reglas de PHP para dar nombre a las matrices son las mismas que las que se usan para las variables regulares: los nombres de las variables deben ir anteceditos con un signo de moneda (\$) y deben iniciar con una letra o un guión bajo, opcionalmente seguidos por más letras, números o guiones bajos. No se permiten signos de puntuación ni espacios en blanco en los nombres de variables de las matrices.

Una vez que hayas decidido el nombre de tu matriz, hay dos maneras de asignarle valores. La primera es el método que viste en la sección anterior, donde los valores son asignados uno por uno y separados por comas. Este método crea una matriz estándar indexada por números. He aquí un ejemplo:

```
<?php
// define una matriz
$carros = array(
    'Ferrari',
    'Porsche',
    'Jaguar',
    'Lamborghini',
    'Mercedes'
);
?>
```

La segunda manera de crear una matriz semejante es estableciendo valores individuales utilizando notaciones indexadas. He aquí un ejemplo, equivalente al anterior:

```
<?php
// define una matriz
$carros[0] = 'Ferrari';
$carros[1] = 'Porsche';
$carros[2] = 'Jaguar';
$carros[3] = 'Lamborghini';
$carros[4] = 'Mercedes';
?>
```

## Pregunta al experto

**P:** En todos estos ejemplos has especificado explícitamente el número de índice de la matriz para cada declaración de asignación. ¿Qué sucede si ignoro cuál es el número de índice correcto? ¿Puedo hacer que PHP asigne de manera automática el siguiente número de índice disponible en la matriz a un nuevo valor?

**R:** Para asignar automáticamente el siguiente número de índice disponible en la matriz, omite los números de índice en la declaración de asignación de la matriz, como se muestra a continuación:

```
<?php
// define una matriz
$carros[] = 'Ferrari';
$carros[] = 'Lamborghini';
?>
```

También puedes utilizar ambas técnicas con matrices asociativas. Para asignar todos los valores de este tipo de matriz en una sola declaración, establece una palabra clave para cada valor y vincula ambas utilizando el conector `=>`; recuerda separar cada par palabra clave-valor con comas. He aquí un ejemplo:

```
<?php
// define una matriz
$datos = array(
    'nombredeusuario' => 'juan',
    'contrasena' => 'secreta',
    'servidor' => '192.168.0.1'
);
?>
```

También puedes asignar valores a palabras clave uno por uno, como en el siguiente ejemplo:

```
<?php
// define una matriz
$datos['nombredeusuario'] = 'juan';
$datos['contrasena'] = 'secreta';
$datos['servidor'] = '192.168.0.1';
?>
```

Para acceder a un valor de la matriz en el script, simplemente utiliza el nombre y el índice/palabra clave en una expresión y PHP lo reemplazará con el valor correspondiente cuando se ejecute el script, igual que en una variable estándar. He aquí un ejemplo de cómo funciona:

```
<?php
// define una matriz
$datos = array(
    'nombredeusuario' => 'juan',
    'contrasena' => 'secreta',
    'servidor' => '192.168.0.1',
);

// usa un valor de la matriz
echo 'La contraseña es: ' . $datos['contrasena'];
?>
```

## NOTA

¿Recuerdas los contenedores especiales de variables llamados `$_GET` y `$_POST` que viste por primera vez en el capítulo 2? Pues bien, ¡también son matrices! Cuando se envía un formulario, PHP convierte automáticamente cada elemento de él en un miembro de la matriz, ya sea `$_GET` o `$_POST`, dependiendo del método de envío.

## Modificar valores de matrices

Modificar el valor de una matriz es tan sencillo como el de cualquier otra variable: simplemente asigna un nuevo valor al elemento utilizando su número de índice o su palabra clave. Examina el siguiente ejemplo, que modifica el segundo elemento de la matriz `$carnes`; cambiará el valor 'jamón' y pondrá en su lugar el valor 'pavo':

```
<?php
// define una matriz
$carnes = array(
    'pescado',
    'pollo',
    'jamón',
    'cordero'
);
```

```
// cambia 'jamón' por 'pavo'
$carnes[2] = 'pavo';
?>
```

Para eliminar un elemento de la matriz, utiliza la función `unset()` en el correspondiente número de índice o palabra clave:

```
<?php
// define una matriz
$carnes = array(
    'pescado',
    'pollo',
    'jamón',
    'cordero'
);

// elimina 'pescado'
unset($carnes[0]);
?>
```

## Pregunta al experto

**P:** Si elimino un elemento en medio de la matriz, ¿qué sucede con los valores anteriores y posteriores?

**R:** Si eliminas un elemento con la función `unset()`, PHP lo declarará nulo (NULL), pero no volverá a indexar automáticamente la matriz. Si se trata de una matriz indexada numéricamente, puedes volver a indexarla, para eliminar los huecos en la secuencia, utilizando la función PHP `array_multisort()` sobre tu matriz.

También puedes eliminar un elemento de una matriz utilizando la función `array_pop()` o `array_push()`; ambas son abordadas más adelante, en este mismo capítulo.

## Recuperar el tamaño de la matriz

Saber cuántos valores contiene una matriz es una tarea importante cuando se trabaja con ellos, en especial cuando se combinan con bucles. Esto se resuelve fácilmente con la función PHP `count()`, que acepta la variable de una matriz como parámetro y regresa un número entero que indica cuántos elementos contiene. He aquí un ejemplo:

```
<?php
// define una matriz
$datos = array('lunes', 'martes', 'miércoles');
```

```
// obtiene el tamaño de la matriz
echo 'La matriz tiene ' . count($datos) . ' elementos';
?>
```

### TIP

En lugar de la función `count()`, puedes utilizar `sizeof()`, que hace exactamente lo mismo.

## Pregunta al experto

**P:** ¿Existe un límite para los elementos que puede contener una matriz en PHP?

**R:** No. La cantidad de elementos en una matriz está limitada por la memoria disponible, como se define en la variable de configuración `'memory_limit'`, en el archivo de configuración de PHP.

## Matrices anidadas

PHP también te permite combinar matrices, colocando una dentro de otra sin límite de profundidad. Esto es útil cuando se trabaja con información ordenada de manera estructurada y jerárquica. Para ilustrarlo, examina el siguiente ejemplo:

```
<?php
// define matrices anidadas
$directorio = array(
    array(
        'nombre' => 'Raymundo Rabbit',
        'tel' => '1234567',
        'correo' => 'ray@planetaconejo.in',
    ),
    array(
        'nombre' => 'David Duck',
        'tel' => '8562904',
        'correo' => 'dduck@lagodepatos.corp',
    ),
    array(
        'nombre' => 'Omar Horse',
        'tel' => '5942033',
        'correo' => 'reyomar@mercadodelgranjero.cosasdecaballos.com',
    )
);
?>
```

En este ejemplo, `$directorio` es una matriz anidada con dos niveles de profundidad. El primero está indexado numéricamente, y cada elemento representa una pieza del directorio telefónico. Cada uno de estos elementos es, a su vez, una matriz asociativa que contiene información específica de los atributos correspondientes a cierta pieza del directorio, como el nombre del contacto, su número telefónico y su dirección de correo electrónico.

Para acceder a un valor que se encuentra en algunos de los niveles internos de la matriz, utiliza la secuencia jerárquica correcta de números de índice y palabras clave para obtenerlo. He aquí un ejemplo, que presenta el número telefónico del contacto 'David Duck':

```
<?php
// define matrices anidadas
$directorio = array(
    array(
        'nombre' => 'Raymundo Rabbit',
        'tel' => '1234567',
        'correo' => 'ray@planetaconejo.in',
    ),
    array(
        'nombre' => 'David Duck',
        'tel' => '8562904',
        'correo' => 'dduck@lagodepatos.corp',
    ),
    array(
        'nombre' => 'Omar Horse',
        'tel' => '5942033',
        'correo' => 'reyomar@mercadodelgranjero.cosasdecaballos.com',
    )
);

// accede al valor anidado
echo "El número telefónico de David Duck es: " . $directorio[1] ['tel'];
?>
```

### **TIP**

¿Quieres mirar dentro de una matriz para ver lo que contiene? Las funciones `var_dump()` y `print_r()` abordadas en el capítulo 2 funcionan tan bien en las matrices como lo hacen en las variables estándar. Compruébalo tú mismo.

## Procesar matrices con bucles e iteradores

Muchas veces tu programa PHP necesitará recorrer una matriz, realizando una operación o un cálculo con cada valor que encuentre. La mejor manera de realizar esta tarea es con bucles, los cuales conociste en el capítulo anterior. Examina el siguiente ejemplo, que establece una matriz y luego realiza operaciones iterativas utilizando el bucle `for`:

```
<?php
// define una matriz
$ciudades = array('Londres', 'París', 'Madrid', 'Los Ángeles', 'Bombay',
'Yakarta');

// hace iteraciones sobre la matriz
// presenta cada valor
for ($i=0; $i<count($ciudades); $i++) {
    echo $ciudades[$i] . "\r\n";
}
?>
```

En este ejemplo, el bucle `for` hace las iteraciones sobre la matriz `$ciudades`, y presenta en pantalla cada valor encontrado. El bucle se ejecuta para cada elemento de la matriz; esta información se averigua rápidamente invocando la función `count()`.

## El bucle `foreach`

Tal vez recuerdes que en el capítulo anterior el tema de los bucles quedó inconcluso, y que aún faltaba por explicar un tipo de ellos. El bucle al que nos referimos es `foreach`, que fue introducido en PHP por primera vez en la versión 4.0 y representa la manera más sencilla de realizar iteraciones sobre matrices, ¡nada sorprendente dado que fue diseñado con ese fin específico!

Con el bucle `foreach`, cada vez que se ejecuta un bucle, el elemento en uso de la matriz es asignado a una variable temporal, la cual puede ser procesada de la forma que más te plazca: mandarlo a pantalla, copiarlo a otra variable, usarlo en cálculos y demás. A diferencia del bucle `for`, `foreach` no utiliza un contador, porque automáticamente “sabe” la posición que ocupa dentro de la matriz en un determinado momento y se desplaza hacia adelante en forma continua hasta que alcanza el final de la matriz; en ese punto se detiene automáticamente.

La mejor manera de comprender esta maravilla de la automatización es con un ejemplo. El siguiente listado lo muestra en acción; es una revisión del ejemplo anterior en que se utiliza el bucle `foreach` en lugar de `for`:

```
<?php
// define una matriz
$ciudades = array('Londres', 'París', 'Madrid', 'Los Ángeles', 'Bombay',
'Yakarta');

// hace iteraciones sobre la matriz
// presenta cada valor
foreach($ciudades as $c) {
    echo "$c \r\n";
}
?>
```



El bucle `foreach` también funciona con matrices asociativas, sólo que en esos casos utiliza dos variables temporales (una para la palabra clave y otra para el valor). El siguiente ejemplo ilustra la diferencia:

```
<?php
// define una matriz
$ciudades = array(
    "Reino Unido" => "Londres",
    "Estados Unidos" => "Washington",
    "Francia" => "París",
    "India" => "Delhi"
);

// hace iteraciones sobre la matriz
// presenta cada valor
foreach($ciudades as $clave => $valor){
    echo "$valor es la capital de $clave. \r\n";
}
?>
```

## El iterador de matrices

Como opción, tal vez quieras utilizar un iterador de matrices llamado precisamente `ArrayIterator` (nuevo en PHP 5.0), que proporciona una herramienta extensible, lista para utilizarse y recorrer en bucle los elementos de una matriz. He aquí un ejemplo:

```
<?php
// define una matriz
$ciudades = array(
    "Reino Unido" => "Londres",
    "Estados Unidos" => "Washington",
    "Francia" => "París",
    "India" => "Delhi"
);

// crea un objeto ArrayIterator
$iterador = new ArrayIterator($ciudades);

// regresa al principio de la matriz
$iterador->rewind();

// hace iteraciones sobre la matriz
// presenta cada valor
while($iterador->valid()){
    print $iterador->current() . " es la capital de " . $iterador->key()
    . ". \r\n";
    $iterador->next();
}
?>
```

En este ejemplo, el objeto `ArrayIterator` se inicializa con una variable de matriz, y el método `rewind()`, perteneciente al objeto, se utiliza para enviar el puntero interno de la matriz al primer elemento de éste. Un bucle `while`, que se ejecuta mientras exista un elemento `valid()`, puede utilizarse entonces para hacer las iteraciones sobre la matriz. Las palabras clave individuales de la matriz son recuperadas por el método `key()`, y sus valores correspondientes son recuperados con el método `current()`. El método `next()` avanza el puntero interno de la matriz al siguiente elemento del mismo.

### NOTA

No te preocupes si la sintaxis utilizada en el objeto `ArrayIterator` no te queda completamente clara. Los objetos son abordados con cierta profundidad en el capítulo 5, y los ejemplos anteriores tendrán mayor sentido conforme avances por el presente capítulo.

## Prueba esto 4-1

## Promediar las calificaciones de un grupo

Ahora que has comprendido los aspectos básicos del trabajo con matrices, hagamos una pequeña aplicación que demuestre su uso práctico. En esta sección escribirás un pequeño script que acepte una matriz de valores; éstos representan las calificaciones numéricas individuales de un grupo de estudiantes de cierto curso, y luego calcularás varias estadísticas de resumen, incluido el promedio general y la cantidad de estudiantes que se encuentran dentro de los rangos del 20% superior y el 20% inferior dentro del grupo.

He aquí el código (*calificaciones.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>Proyecto 4-1: Promedio de Desempeño</title>
    </head>
    <body>
        <h2>Proyecto 4-1: Promedio de Desempeño</h2>
    <?php
    // define una matriz de calificaciones
    // que van de 1 a 100
    $calificaciones = array(
        25, 64, 23, 87, 56, 38, 78, 57, 98, 95,
        81, 67, 75, 76, 74, 82, 36, 39,
        54, 43, 49, 65, 69, 69, 78, 17, 91
    );

    // obtiene la cantidad de calificaciones
    $cuenta = count($calificaciones);
```

(continúa)

```
// hace iteraciones sobre la matriz
// calcula el total y el 20% superior e inferior
$total = $sup = $inf = 0;
foreach ($calificaciones as $g){
    $total += $g;
    if($g <= 20){
        $inf++;
    }

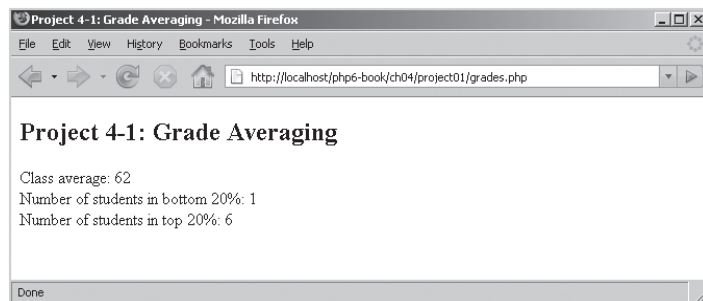
    if ($g >= 80){
        $sup++;
    }
}

// calcula el promedio
$prom = round($total / $cuenta);

// presenta en pantalla las estadísticas
echo "Promedio del grupo: $prom <br />";
echo "Número de estudiantes en el 20% inferior: $inf <br />";
echo "Número de estudiantes en el 20% superior: $sup <br />";
?>
</body>
</html>
```

Este script comienza definiendo una matriz, que contiene las calificaciones de cada estudiante en un curso. Luego, un bucle `foreach` hace las iteraciones sobre la matriz, con lo que genera un acumulativo total; este total es dividido entre el número de estudiantes y es recuperado por la función `count()`, para calcular el promedio del grupo.

Como las calificaciones van del 1 al 100, no es difícil calcular la cantidad de estudiantes que se encuentran en el 20% superior e inferior del grupo. Dentro del bucle `foreach` cada valor es verificado para probar si es menor a 20 o superior a 80 y sus respectivos contadores se incrementan de acuerdo con ello. De esta manera, el resumen de los datos se presenta en una página Web, como se muestra en la figura 4-1.



**Figura 4-1** Las estadísticas de resumen generadas por el programa que promedia el desempeño del grupo

## Utilizar matrices con formularios

Las matrices son muy poderosas cuando se combinan con elementos de formularios Web que dan soporte a más de un valor, como listas de selección múltiple o casillas de verificación agrupadas. Para capturar en una matriz los datos proporcionados por el usuario, simplemente añade un juego de corchetes al elemento 'name' del formulario para convertirlo automáticamente en una matriz de PHP cuando se envíe.

La manera más fácil de ilustrarlo es con un ejemplo. Examina el siguiente formulario, que contiene una lista de selección múltiple con nombres de músicos populares:

```
<form method="post" action="matriz-formulario.php">
  Selecciona el nombre de tu artista favorito: <br />
  <select name="artistas[]" multiple="true">
    <option value="Britney Spears">Britney Spears</option>
    <option value="Aerosmith">Aerosmith</option>
    <option value="Black-Eyed Peas">Black-Eyed Peas</option>
    <option value="Diana Ross">Diana Ross</option>
    <option value="Foo Fighters">Foo Fighters</option>
  </select>
  <p>
    <input type="submit" name="submit" value="Enviar" />
  </form>
```

Pon atención al atributo 'name' del elemento <select>, que tiene el nombre `artistas[]`. Esto le indica a PHP que, cuando se envíe el formulario, todos los valores seleccionados de la lista deben transformarse en elementos de una matriz. El nombre de ésta será `$_POST['artistas']`, y tendrá más o menos este aspecto:

```
Array
(
    [0] => Britney Spears
    [1] => Black-Eyed Peas
    [2] => Diana Ross
)
```

### Prueba esto 4-2 Seleccionar sabores de pizzas

Una aplicación práctica hará que sea más fácil comprender la manera en que PHP interactúa con variables de matrices y formularios. La siguiente aplicación presenta al usuario un formulario que contiene varios sabores de pizzas y le solicita que seleccione sus favoritas, mediante casillas de verificación. He aquí el formulario (*pizza.html*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
```

(continúa)

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 4-2: Selector de Pizzas Favoritas</title>
  </head>
  <body>
    <h2>Proyecto 4-2: Selector de Pizzas Favoritas</h2>
    <form method="post" action="pizza.php">
      Seleccione su pizza favorita: <br />
      <input type="checkbox" name="favorita[]" value="tomate">Tomate</
input>
      <input type="checkbox" name="favorita[]" value="cebolla">Cebolla</
input>
      <input type="checkbox" name="favorita[]" value="jalapeños">Jalapeño
s</input>
      <input type="checkbox" name="favorita[]" value="aceitunas">Aceituna
s</input>
      <input type="checkbox" name="favorita[]" value="suiza">Suiza</
input>
      <input type="checkbox" name="favorita[]" value="piña">Piña</input>
      <input type="checkbox" name="favorita[]" value="tocino">Tocino</
input>
      <input type="checkbox" name="favorita[]" value="pollo">Pollo</
input>
      <input type="checkbox" name="favorita[]" value="jamón">Jamón</
input>
      <input type="checkbox" name="favorita[]" value="anchoas">Anchoas</
input>
      <input type="checkbox" name="favorita[]" value="extraqueso">Extra
queso</input>
      <p>
        <input type="submit" name="submit" value="Enviar" />
      </p>
    </form>
  </body>
</html>
```

La figura 4-2 muestra cómo aparece el formulario.

Y a continuación, el código del script que capta los datos del formulario (*pizza.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 4-2: Selector de Pizzas Favoritas</title>
  </head>
  <body>
    <h2>Proyecto 4-2: Selector de Pizzas Favoritas</h2>
    Usted ha seleccionado las siguientes pizzas como sus favoritas: <br />
    <ul>
```



**Figura 4-2** Un formulario Web para seleccionar pizzas favoritas

```
<?php
foreach ($_POST['favorita'] as $f) {
    echo "<li>$f</li> \r\n";
}
?>
</ul>
</body>
</html>
```

Cuando se envía el formulario Web, automáticamente PHP coloca todos los valores seleccionados en la matriz `$_POST['favorita']`. Esta matriz puede procesarse como cualquiera otra, en este caso con un bucle `foreach` que presenta en pantalla las pizzas seleccionadas en una lista. La figura 4-3 muestra los datos de salida.



**Figura 4-3** El resultado del formulario enviado

# Trabajar con funciones de matrices

PHP cuenta con numerosas funciones integradas para manipular matrices; da soporte a operaciones que van desde la búsqueda y comparación, hasta la organización y conversión de elementos dentro de la matriz. La tabla 4-1 muestra algunas de esas funciones.

## Conversión entre cadenas de caracteres y matrices

PHP te permite convertir una cadena de caracteres en una matriz, al tratar la cadena con separadores definidos por el usuario y asignar los segmentos resultantes a una matriz. La función

Función	Lo que hace
<code>explode()</code>	Divide una cadena de caracteres en elementos de una matriz
<code>implode()</code>	Conjunta elementos de una matriz en una cadena de caracteres
<code>range()</code>	Genera un rango de números como matriz
<code>min()</code>	Encuentra el valor menor dentro de la matriz
<code>max()</code>	Encuentra el valor mayor dentro de la matriz
<code>shuffle()</code>	Reordena aleatoriamente una secuencia de elementos dentro de la matriz
<code>array_slice()</code>	Extrae un segmento de la matriz
<code>array_shift()</code>	Elimina un elemento del principio de la matriz
<code>array_unshift()</code>	Añade un elemento al principio de la matriz
<code>array_pop()</code>	Elimina un elemento del final de la matriz
<code>array_push()</code>	Añade un elemento al final de la matriz
<code>array_unique()</code>	Elimina elementos repetidos dentro de la matriz
<code>array_reverse()</code>	Invierte la secuencia de los elementos dentro de la matriz
<code>array_merge()</code>	Combina dos o más matrices
<code>array_intersect()</code>	Calcula los elementos comunes entre dos o más matrices
<code>array_diff()</code>	Calcula las diferencias entre dos matrices
<code>in_array()</code>	Verifica si un valor existe dentro de la matriz
<code>array_key_exists()</code>	Verifica si una clave en particular existe dentro de la matriz
<code>sort()</code>	Ordena una matriz
<code>asort()</code>	Ordena una matriz asociativa por valor
<code>ksort()</code>	Ordena una matriz asociativa por palabra clave
<code>rsort()</code>	Revierde el orden de una matriz
<code>krsort()</code>	Revierde el orden de una matriz asociativa por valor
<code>arsort()</code>	Revierde el orden de una matriz asociativa por palabra clave

**Tabla 4-1** Funciones de matriz comunes en PHP

PHP que realiza esta tarea lleva el nombre de `explode()`, acepta dos argumentos: el separador y la cadena de caracteres fuente, y regresa una matriz. He aquí un ejemplo:

```
<?php
// define una cadena de caracteres
$cadena = 'policía,sastre,soldado,espía';

// convierte una cadena en matriz
// datos de salida: ('policía', 'sastre', 'soldado', 'espía')
$matriz = explode(',', $cadena);
print_r($matriz);
?>
```

También es posible revertir el proceso, conjuntar elementos de una matriz en una sola cadena de caracteres utilizando el “pegamento” proporcionado por el usuario. La función PHP para realizar esta tarea recibe el nombre de `implode()` y se muestra en el siguiente ejemplo:

```
<?php
// define una matriz
$matriz = array('uno', 'dos', 'tres', 'cuatro');

// convierte matriz en cadena de caracteres
// datos de salida: 'uno y dos y tres y cuatro'
$cadena = implode(' y ', $matriz);
print_r($cadena);
?>
```

## Trabajar con rangos de números

Si tratas de llenar una matriz con un rango de números, la función `range()` es una mejor opción que ingresar manualmente cada valor. Esta función acepta dos extremos y regresa una matriz que contiene todos los números existentes entre los extremos establecidos. He aquí un ejemplo; genera una matriz que contiene todos los valores entre 1 y 1000:

```
<?php
// define una matriz
$matriz = range(1,1000);
print_r($matriz);
?>
```

Como opción, si ya cuentas con una matriz de números y quieres calcular el mínimo y el máximo de la serie, las funciones PHP `min()` y `max()` serán de utilidad; aceptan una matriz



numérica y regresan el valor menor y mayor, respectivamente, de los elementos contenidos en la matriz. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz
$matriz = array(7, 36, 5, 48, 28, 90, 91, 3, 67, 42);

// obtiene mínimos y máximos
// datos de salida: 'El mínimo es 3 y el máximo es 91'
echo 'El mínimo es ' . min($matriz) . ' y el máximo es ' . max($matriz);
?>
```

## Extraer segmentos de la matriz

PHP te permite cortar una matriz en partes pequeñas con la función `array_slice()`, que acepta tres argumentos: la matriz original, la posición del índice (*offset*) donde debe comenzar el corte y la cantidad de elementos que debe regresar a partir de la posición de inicio. El siguiente ejemplo ilustra esta acción:

```
<?php
// define una matriz
$arcoiris = array('violeta', 'índigo', 'azul', 'verde', 'amarillo',
    'naranja', 'rojo');

// extrae 3 valores centrales
// datos de salida: ('azul', 'verde', 'amarillo')
$matriz = array_slice($arcoiris, 2, 3);
print_r($matriz);
?>
```

Para extraer un segmento a partir del final de la matriz (en lugar de hacerlo desde el principio), inserta un valor negativo para la posición del índice en la función `array_slice()`, como se presenta en la siguiente revisión del ejemplo anterior:

```
<?php
// define una matriz
$arcoiris = array('violeta', 'índigo', 'azul', 'verde', 'amarillo',
    'naranja', 'rojo');

// extrae 3 valores centrales
// a partir del final
// datos de salida: ('azul', 'verde', 'amarillo')
$matriz = array_slice($arcoiris, -5, 3);
print_r($matriz);
?>
```

## Añadir y eliminar elementos de la matriz

PHP contiene cuatro funciones que te permiten añadir o eliminar elementos del principio o el final de la matriz: `array_unshift()` añade un elemento al principio; `array_shift()` elimina el primer elemento; `array_push()` añade un elemento al final; `array_pop()` elimina el último elemento de la matriz. El siguiente ejemplo los muestra en acción:

```
<?php
// define una matriz
$filmes = array('El Rey León', 'Cars', 'Bichos');

// elimina el primer elemento de la matriz
array_shift($filmes);

// elimina el último elemento de la matriz
array_pop($filmes);

// añade un elemento al final de la matriz
array_push($filmes, 'Ratatouille');

// añade un elemento al principio de la matriz
array_unshift($filmes, 'Los Increíbles');

// muestra la matriz
// datos de salida: ('Los Increíbles', 'Cars', 'Ratatouille')
print_r($filmes);
?>
```

### **NOTA**

Las funciones `array_unshift()`, `array_shift()`, `array_push()` y `array_pop()` sólo deben utilizarse con matrices indexadas numéricamente y no con asociativas. Cada una de estas funciones indexa de nuevo la matriz para llevar la cuenta del valor o los valores añadidos o eliminados durante su operación.

## Eliminar elementos duplicados en la matriz

PHP te permite limpiar una matriz de valores duplicados con la función `array_unique()`, que acepta la matriz completa y regresa una nueva que sólo contiene valores únicos. El siguiente ejemplo la muestra en acción:

```
<?php
// definir una matriz
$duplicados = array('a', 'b', 'a', 'c', 'e', 'd', 'e');
```

```
// elimina duplicados
// datos de salida: ('a', 'b', 'c', 'e', 'd')
$originales = array_unique($duplicados);
print_r($originales);
?>
```

## Ordenar aleatoriamente e invertir la matriz

La función PHP `shuffle()` transforma el orden actual de los elementos de la matriz para darles un orden aleatorio, mientras que la función `array_reverse()` invierte el orden de sus elementos. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz
$arcoiris = array('violeta', 'índigo', 'azul', 'verde', 'amarillo',
    'naranja', 'rojo');

// da orden aleatorio a la matriz
shuffle($arcoiris);
print_r($arcoiris);

// invierte los elementos de la matriz
// datos de salida: ('rojo', 'naranja', 'amarillo', 'verde', 'azul',
// 'índigo', 'violeta')
$matriz = array_reverse($arcoiris);
print_r($matriz);
?>
```

## Realizar búsquedas en la matriz

La función `in_array()` revisa la matriz en busca de un valor específico y regresa una respuesta verdadera (true) en caso de localizarlo. He aquí un ejemplo, que busca la cadena de caracteres 'Barcelona' en la matriz `$ciudades`:

```
<?php
// define una matriz
$ciudades = array('Londres', 'París', 'Barcelona', 'Lisboa', 'Zurich');

// busca un valor dentro de la matriz
echo in_array('Barcelona', $ciudades);
?>
```

Si en lugar de valores quieres buscar palabras clave de una matriz asociativa, PHP también puede realizar esa acción: la función `array_key_exists()` busca una coincidencia entre las palabras clave de la matriz y el término específico que se busca. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz
$ciudades = array(
    "Reino Unido" => "Londres",
    "Estados Unidos" => "Washington",
    "Francia" => "París",
    "India" => "Delhi"
);

// busca palabra clave en la matriz
echo array_key_exists('India', $ciudades);
?>
```

## Ordenar matrices

PHP cuenta con varias funciones integradas diseñadas para ordenar matrices de muy diversas maneras. La primera es la función `sort()`, que te permite disponer las matrices indexadas numéricamente por orden alfabético o numérico, de menor a mayor. He aquí un ejemplo:

```
<?php
// define una matriz
$datos = array(15, 81, 14, 74, 2);

// ordena y presenta la matriz
// datos de salida: (2, 14, 15, 74, 81)
sort($datos);
print_r($datos);
?>
```

Sin embargo, cuando quieres organizar una matriz asociativa, es mejor utilizar la función `asort()`, que mantiene la correlación entre palabras clave y valores mientras realiza la organización. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz
$perfil = array(
    "nombre" => "Susana",
    "apellido" => "De Tal",
    "sexo" => "femenino",
    "sector" => "Administración de recursos"
);

// ordena por valor
// datos de salida: ('sector' => 'Administración de recursos'
// 'apellido' => 'De Tal',
```

```
// 'nombre' => 'Susana'
// 'sexo' => 'femenino')
asort($perfil);
print_r($perfil);
?>
```

La función `ksort()` también se aplica a matrices asociativas; utiliza las palabras clave en lugar de los valores para ordenar la matriz. He aquí un ejemplo:

```
<?php
// define una matriz
$perfil = array(
    "nombre" => "Susana",
    "apellido" => "De Tal",
    "sexo" => "femenino",
    "sector" => "Administración de recursos"
);

// organiza por palabra clave
// datos de salida: ('apellido' => 'De Tal',
// 'nombre' => 'Susana',
// 'sector' => 'Administración de recursos',
// 'sexo' => 'femenino')
ksort($perfil);
print_r($perfil);
?>
```

### ***TIP***

Para invertir la secuencia ordenada que generaron `sort()`, `asort()` y `ksort()`, utiliza las funciones `rsort()`, `arsort()` y `krsort()`, respectivamente.

## Combinar matrices

PHP te permite combinar dos o más matrices con la función `array_merge()`, que acepta variables de una o más matrices. El siguiente ejemplo y los datos de salida muestran su uso:

```
<?php
// define matrices
$oscuro = array('negro', 'café', 'azul');
$claro = array('blanco', 'plateado', 'amarillo');

// combina matrices
// datos de salida: ('negro', 'café', 'azul',
//                  'blanco', 'plateado', 'amarillo')
$colores = array_merge($oscuro, $claro);
print_r($colores);
?>
```

## Comparar matrices

PHP proporciona dos funciones para comparar matrices: `array_intersect()`, que regresa los valores comunes entre dos matrices y `array_diff()` que regresa los valores de la primera matriz que no existen en la segunda. He aquí un ejemplo que ilustra ambas en acción:

```
<?php
// define matrices
$naranja = array('rojo', 'amarillo');
$verde = array('amarillo', 'azul');

// encuentra elementos comunes
// datos de salida: ('amarillo')
$comunes = array_intersect($naranja, $verde);
print_r($comunes);

// encuentra elementos de la primera matriz que no están en la segunda
// datos de salida: ('rojo')
$unico = array_diff($naranja, $verde);
print_r($unico);
?>
```

### TIP

También puedes comparar matrices con el operador de comparación de PHP (`==`), de manera muy similar a la comparación de variables.

## Prueba esto 4-3

## Verificar números primos

Ahora que has visto las muchas maneras en que PHP te permite trabajar con matrices, hagamos un pequeño ejemplo práctico que muestre algunas de estas funciones integradas en acción. La siguiente aplicación solicita al usuario que ingrese una serie de números en un formulario Web, y regresa un mensaje indicando cuáles son números primos.

He aquí el código (*primos.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>Proyecto 4-3: Prueba de Números Primos</title>
    </head>
    <body>
        <h2>Proyecto 4-3: Prueba de Números Primos</h2>
```

(continúa)

```
<?php
// si el formulario no ha sido enviado
// presenta el formulario
if (!isset ($_POST['submit'])) {
?>
<form method="post" action="primos.php">
    Escriba una lista de números, separados por comas: <br />
    <input type='text' name='num' />
    <p>
        <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
// si el formulario ya ha sido enviado
// procesa los datos de entrada
} else {
    // recupera los números del envío POST
    // los convierte en una matriz de acuerdo con la separación por
comas
    $cadNums = $_POST['num'];
    $matrizNums = explode(',', $_POST['num']);
    $primos = array();
    $primosMarca = 0;

    // itera sobre la matriz
    // obtiene el valor absoluto de cada número
    foreach ($matrizNums as $n){
        $n = trim(abs ($n));

        // verifica si cada número es primo:
        // prueba cada número dividiéndolo
        // por todos los números existentes entre él mismo y el 2
        // si no es perfectamente divisible por ninguno
        // el número es primo
        for ($i=2; $i<$n; $i++){
            $primosMarca = 0;
            if (($n%$i) == 0){
                break;
            }
            $primosMarca = 1;
        }

        // si es primo
        // se añade a la matriz de salida
        if ($primosMarca == 1){
            array_push($primos, $n);
        }
    }
}
```

```

    }
    // verifica si se encontraron números primos
    // de ser así, los organiza y elimina duplicados de la matriz
    // presenta el mensaje en pantalla
    if (count ($primos) > 0) {
        $primos = array_unique($primos);
        sort ($primos);
        echo 'Los siguientes números son primos: ' . implode($primos, ' ');
    } else {
        echo 'No se encontraron números primos';
    }
}
?>
</body>
</html>

```

Utilizando una técnica que ahora debe serte familiar, esta aplicación combina un formulario Web con la página de resultados en un mismo script, separando ambas con la declaración condicional `if-else`. El formulario Web permite que el usuario ingrese una serie de números separados por comas. La figura 4-4 presenta cómo se ve el formulario en el explorador Web.

Una vez que el usuario ingresa los números y envía el formulario, se dispara la segunda parte del script. Debe ser claro que esta parte del script utiliza muchas de las funciones de matrices abordadas en la sección anterior. Primero, la función `explode()` se encarga de extraer los números individuales enviados por el usuario y colocarlos en una matriz, utilizando



**Figura 4-4** Formulario Web para ingresar los números

(continúa)





---

**Figura 4-5** Los resultados del formulario enviado, que muestran los números primos encontrados

la coma como separador. A continuación, un bucle `foreach` itera sobre la matriz, calculando primero el valor absoluto de cada número y luego dividiéndolo entre todos los números existentes entre él y el 2 para determinar si es un número primo o no.

Si un número no se puede dividir por lo menos entre otro número diferente de sí mismo o de la unidad, sin que quede un residuo, se le considera primo, y se utiliza la función `array_push()` para añadirlo como elemento a una nueva matriz llamada `$primos`. Una vez que todos los números enviados por el usuario pasan por este proceso, se utiliza la función `count()` para averiguar si se encontraron números primos en el proceso; de ser así, la matriz `$primos` pasa por una verificación que elimina los duplicados, ordena los valores de menor a mayor y envía los resultados a la página como datos de salida.

La figura 4-5 muestra los datos de salida obtenidos de la secuencia enviada.

---

## Trabajar con fechas y horas

Además de la capacidad de almacenar múltiples valores en una matriz, PHP también cuenta con un conjunto completo de funciones para trabajar con fechas y horas. Aunque el lenguaje no cuenta con un tipo de datos dedicado exclusivamente a valores de fecha y hora, ofrece a los programadores gran flexibilidad para darles formato y manipularlos.

## Pregunta al experto

**P:** ¿Qué es el sello cronológico (timestamp) de UNIX?

**R:** Muy sencillo, el valor de sello cronológico de UNIX para cualquier punto en el tiempo, representa la cantidad de segundos que han transcurrido desde la medianoche del 1 de enero de 1970 y el momento actual. De esa manera, por ejemplo, el punto del tiempo ubicado el 5 de enero de 2008 a las 10:15:00 a.m. se representa en formato de sello cronológico de UNIX con el formato 1199508300.

PHP puede convertir automáticamente un valor de fecha en el formato de sello cronológico de UNIX con la función `mktime()`, que acepta argumentos para día, mes, año, hora, minuto y segundo, para regresar el sello cronológico correspondiente a ese instante.

## Generar fechas y horas

Como muchos otros lenguajes de programación, PHP representa valores de fecha/hora en el formato de sello cronológico de UNIX. La generación del sello cronológico se realiza generalmente con la función `mktime()`, que acepta una serie de parámetros de fecha y hora convencionales para convertirlos en su correspondiente valor de sello cronológico. Para ilustrarlo, examina el siguiente ejemplo, que regresa el sello cronológico correspondiente al 5 de enero de 2008 a las 10:15:00 a.m.

```
<?php
// regresa el sello cronológico del 5 de enero de 2008 a las 10:15
// datos de salida: 1199508300
echo mktime(10,15,00,1,5,2008);
?>
```

La invocación de la función `mktime()` sin argumentos regresa el sello cronológico de UNIX correspondiente a la hora actual:

```
<?php
// regresa el sello cronológico del momento actual
echo mktime();
?>
```

Otra forma de obtener la fecha y hora actuales es con la función PHP `getdate()`, que regresa una matriz asociativa que contiene información sobre la fecha y hora actuales. He aquí un ejemplo de esa matriz:

```
Array
(
    [seconds] => 33
    [minutes] => 27
```

```
[hours] => 19
[mday] => 12
[wday] => 1
[mon] => 11
[year] => 2007
[yday] => 315
[weekday] => Monday
[month] => November
[0] => 1194875853
)
```

He aquí un ejemplo de esta función en acción:

```
<?php
// obtiene la fecha y horas actuales como una matriz
$hoy = getdate();

// datos de salida: 'Fecha y hora actual 19:26:23 del 12-11-2007'
echo 'Fecha y hora actual ' . $hoy ['hours'] . ':' . $hoy ['minutes'] .
':' . $hoy ['seconds'] . ' del ' . $hoy ['mday'] . '-' . $hoy ['mon'] .
'-' . $hoy ['year'];
?>
```

### ***TIP***

Advierte que la matriz regresada por `getdate()` incluye la representación de la fecha en formato de sello cronológico de UNIX en la posición 0 del índice.

## Formar fechas y horas

Casi siempre, generar el sello cronológico es sólo la mitad de la batalla: también necesitas encontrar una forma de presentarlo que sea legible para los humanos. Ahí es donde la función PHP `date()` entra en juego: esta función te permite transformar la horrible y larga cadena de sello cronológico en algo mucho más informativo. Esta función acepta dos argumentos: una cadena de formato y el valor de sello cronológico. La cadena de formato está conformada por una secuencia de caracteres, cada uno de los cuales tiene un significado especial. La tabla 4-2 presenta una lista con los más utilizados.

Utilizando los caracteres especiales de la tabla 4-2, es posible dar formato al sello cronológico de UNIX con la función `date()`, para personalizar la manera en que los valores de fecha y hora son presentados en pantalla. He aquí algunos ejemplos:

```
<?php
// datos de salida: "Hora y fecha actual: 12:28 pm 20 Mar 2008"
echo ' Hora y fecha actual: ' . date("h:i a d M Y",
mktime(12,28,13,3,20,2008));
```

Carácter	Significado
d	Día del mes (numérico)
D	Día de la semana (cadena de caracteres)
l	Día de la semana (cadena de caracteres)
F	Mes (cadena de caracteres)
M	Mes (cadena de caracteres)
m	Mes (numérico)
Y	Año
h	Hora (en formato de 12 horas)
H	Hora (en formato de 24 horas)
a	a.m. o p.m.
i	Minuto
s	Segundo

**Tabla 4-2** Códigos de formato para la función `date()`

```
// datos de salida: " Hora y fecha actual: 8:15 14 Feb 2008"
echo ' Hora y fecha actual: ' . date("H:i d F Y",
mktime(8,15,0,2,14,2008));

// datos de salida: "La fecha de hoy es Oct-05-2007"
echo 'La fecha de hoy es ' . date("M-d-Y", mktime(0,0,0,10,5,2007));
?>
```

## Funciones de fecha y hora útiles

PHP también da soporte a muchas otras funciones de manipulación para fechas y horas, que te permiten verificar si la fecha es válida o hacer conversiones entre zonas horarias. La tabla 4-3 presenta algunas de ellas.

Función	Lo que hace
<code>checkdate()</code>	Verifica si es una fecha válida
<code>strtotime()</code>	Crea sellos cronológicos para descripciones en inglés
<code>gmdate()</code>	Expresa un sello cronológico en GMT

**Tabla 4-3** Funciones comunes de fecha y hora de PHP

## Validar una fecha

Una función integrada de gran utilidad es `checkdate()`, que acepta combinaciones de mes, día y año y regresa un valor *true/false* (verdadero/falso) indicando si la fecha es válida o no. Examina el siguiente ejemplo, que prueba la fecha 30 de febrero de 2008.

```
<?php
// datos de salida: 'Fecha no válida'
if(checkdate(2,30,2008)) {
    echo 'Fecha válida';
} else {
    echo 'Fecha no válida';
}
?>
```

## Convertir cadenas de caracteres en sellos cronológicos

Otra función PHP de gran utilidad es `strtotime()`, que acepta una cadena de caracteres que contenga una fecha o una hora y la convierte en un sello cronológico de UNIX. He aquí un ejemplo:

```
<?php
// define una cadena de caracteres que contiene un valor de fecha
// la convierte en un sello cronológico de UNIX
// aplica formato utilizando date()
// datos de salida: '07 Jul 08'
$cadena = 'July 7 2008';
echo date('d M y', strtotime($cadena));
?>
```

Resulta interesante apuntar que la función `strtotime()` reconoce descripciones familiares de tiempo como "now" (ahora), "3 hours ago" (hace tres horas), "tomorrow" (mañana) o "next Friday" (el próximo viernes). El siguiente ejemplo ilustra tan útil característica:

```
<?php
// datos de salida: '12 Mar 09'
echo date('d M y', strtotime('now'));

// datos de salida: '13 Mar 09'
echo date('d M y', strtotime('tomorrow'));

// datos de salida: '16 Mar 09'
echo date('d M y', strtotime('next Friday'));
```

```
// datos de salida: '10 Mar 09'
echo date ('d M y', strtotime('48 hours ago'));
?>
```

## Traducir números de día y mes a nombres

La función `date()` abordada en la sección anterior no sólo es útil para dar formato a los sellos cronológicos y convertirlos en cadenas legibles; también sirve para encontrar el día de la semana correspondiente a una fecha determinada. Para ello, simplemente utiliza el carácter de formato `'D'` con el sello cronológico, como se ilustra en el siguiente ejemplo, que muestra el día de la semana correspondiente al 5 de octubre de 2008:

```
<?php
// datos de salida: 'Sun'
echo date ('D', mktime(0,0,0,10,5,2008));
?>
```

También puedes hacerlo para los nombres de los meses, utilizando el parámetro `'F'` de la función `date()`:

```
<?php
// muestra una lista con los nombres de los meses
// datos de salida: 'January, February, ... December'
foreach (range(1,12) as $m){
    $meses[] = date('F', mktime(0,0,0,$m,1,0));
}
echo implode($meses, ', ');
?>
```

## Calcular la hora GMT (hora del Meridiano de Greenwich)

La función `gmdate()` trabaja exactamente como `date()`, salvo que a partir de la cadena con datos de fecha regresa la hora GMT en lugar de la hora local. Para verlo en acción, examina los siguientes ejemplos, que regresan el GMT equivalente a dos horas locales:

```
<?php
// muestra el GTM relativo a 'now'
echo gmdate('H:i:s d-M-Y', mktime());

// muestra el GMT relativo a '18:01 30-Nov-2007'
// datos de salida: '00:01:00 01-Dec-2007'
echo gmdate('H:i:s d-M-Y', mktime(18,1,0,11,30,2007));
?>
```

## Prueba esto 4-4 Construir una calculadora de edad

Ahora que sabes un poco sobre la manera en que PHP maneja las fechas y los horarios, pongamos este conocimiento en uso con un proyecto práctico: una aplicación Web que permite escribir tu fecha de nacimiento y calcula la edad que tienes en este momento, en años y meses.

He aquí el código (*calculaedad.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml11-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 4-4: Calculadora de Edades</title>
  </head>
  <body>
    <h2>Proyecto 4-4: Calculadora de Edades</h2>
  <?php
    // si el formulario no ha sido enviado
    // muestra el formulario
    if (!isset ($_POST['submit'])) {
  ?>
    <form method="post" action="calculaedad.php">
      Escribe tu fecha de nacimiento, en formato mm/dd/aaaa: <br />
      <input type="text" name="fdn" />
      <p>
        <input type="submit" name="submit" value="Enviar" />
      </form>
  <?php
    // si el formulario ha sido enviado
    // procesa los datos enviados
    } else {
      // divide el valor de la fecha en sus componentes
      $fechaArr = explode('/', $_POST['fdn']);

      // calcula el sello cronológico correspondiente al valor de la fecha
      $fechaTs = strtotime($_POST['fdn']);

      // calcula el sello cronológico correspondiente al día de hoy, 'today'
      $now = strtotime('today');

      // verifica si los datos han sido enviados con el formato correcto
      if (sizeof($fechaArr) != 3) {
        die('ERROR: Por favor escriba una fecha válida');
      }
      // verifica si los datos insertados son una fecha válida
      if (!checkdate($fechaArr[0], $fechaArr[1], $fechaArr[2])) {
```

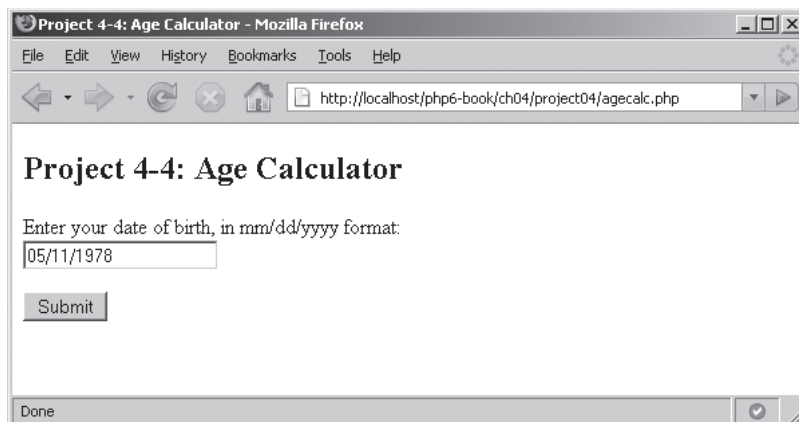
```

        die ('ERROR: Por favor escriba una fecha de nacimiento válida');
    }
    // verifica que la fecha sea anterior a hoy, 'today'
    if($fechaTs >= $now) {
        die('ERROR: Por favor escriba una fecha anterior al día de hoy');
    }

    // calcula la diferencia entre la fecha de nacimiento y el día de
    hoy en días
    // convierte en años
    // convierte los días restantes en meses
    // presenta los datos de salida
    $edadDias = floor(($now - $fechaTs) / 86400);
    $edadAnos = floor($edadDias / 365);
    $edadMeses = floor(($edadDias - ($edadAnos * 365)) / 30);
    echo "Su edad aproximada es $edadAnos años y $edadMeses meses.";
}
?>
</body>
</html>

```

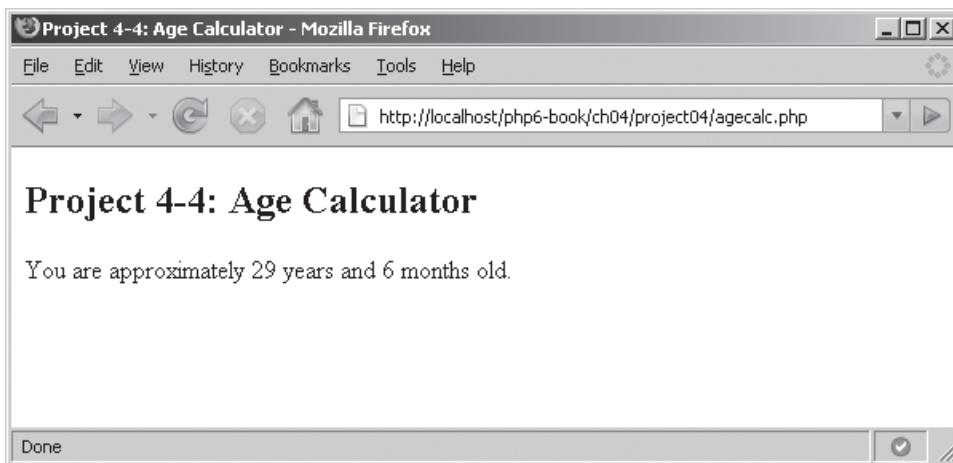
Una vez que el usuario envía su fecha de nacimiento en el formulario Web (figura 4-6), siguiendo el formato MM/DD/AAAA, se dispara la segunda parte del script. La primera parte del programa se concentra especialmente en buscar errores. Primero, se divide la fecha ingresada en el formulario en sus valores constituyentes de mes, día y año; la función `checkdate()` se utiliza para verificar si se trata de una fecha válida. A continuación, la función `strtotime()` se utiliza para convertir la fecha ingresada en sello cronológico de UNIX; la cadena que surge es comparada con el valor “today” (hoy) para verificar que se trata de una fecha pasada.



**Figura 4-6** Formulario Web para que el usuario escriba su edad

(continúa)





**Figura 4-7** El resultado del formulario enviado muestra el cálculo de la edad

Dando por hecho que la fecha enviada aprueba la revisión de errores, el script pasa a realizar los cálculos. Primero, el sello cronológico correspondiente a “today” (hoy) se resta del sello correspondiente a la fecha de nacimiento; como ambos valores están expresados en segundos, el resultado de la operación regresa la edad del usuario en segundos. Este resultado es dividido entre 86 400, la cantidad de segundos que tiene un día, y regresa la edad del usuario en días. Al dividir el número de días entre 365 se obtiene la edad del usuario en años. Por último, la parte restante de la edad (meses) se calcula dividiendo los días restantes entre 30.

La figura 4-7 muestra el resultado después de que se envía el formulario.

## Resumen

Este capítulo te presentó un nuevo tipo de variable de PHP, las matrices, que te permiten agrupar múltiples variables relacionadas y trabajar en ellas como una sola entidad. Además de enseñarte a crear, editar y procesar matrices, también te mostró algunas de las funciones de PHP para manipularlas, te enseñó la manera de utilizarlas en el contexto de los formularios Web y pusiste en práctica las matrices mediante tres aplicaciones sencillas. Por último, te explicé cómo trabajar con datos de tiempo en PHP, enseñándote las funciones de fecha y hora más comunes, que fueron utilizadas para construir una sencilla calculadora de edades.

Las matrices son una importante adición a tu caja de herramientas; a medida que tus scripts se vuelvan más complejos, comenzarás a apreciar el poder y la flexibilidad de las matrices. Para aprender más al respecto, puedes visitar las siguientes direcciones Web:

- Matrices, en [www.php.net/manual/en/language.types.array.php](http://www.php.net/manual/en/language.types.array.php)
- Operadores de matrices, en [www.php.net/manual/en/language.operators.array.php](http://www.php.net/manual/en/language.operators.array.php)
- Funciones para manipulación de matrices, en [www.php.net/manual/en/ref.array.php](http://www.php.net/manual/en/ref.array.php)
- Los matrices especiales \$\_POST y \$\_GET, en [www.php.net/manual/en/reserved.variables.php#reserved.variables.get](http://www.php.net/manual/en/reserved.variables.php#reserved.variables.get)
- Funciones de fecha y hora, en [www.php.net/manual/en/ref.datetime.php](http://www.php.net/manual/en/ref.datetime.php)

## Autoexamen Capítulo 4

1. ¿Cuáles son los dos tipos de matriz PHP y en qué se diferencian?
2. Menciona las funciones que utilizarías para realizar las siguientes tareas:
  - A** Eliminar elementos duplicados en una matriz
  - B** Añadir un elemento al inicio de la matriz
  - C** Invertir el orden de una matriz
  - D** Contar la cantidad de elementos de una matriz
  - E** Buscar un valor dentro de una matriz
  - F** Mostrar el contenido de una matriz
  - G** Revolver el contenido de una matriz
  - H** Combinar dos matrices en una sola
  - I** Encontrar los elementos comunes entre dos matrices
  - J** Convertir una cadena de caracteres en una matriz
  - K** Extraer un segmento de una matriz
3. ¿Cuáles serían los datos de salida del siguiente código?

```
<?php
sort(array_slice(array_reverse(array_unique(array('b','i','g','f','o',
'o','t'))), 2, 3));
?>
```

4. Utilizando únicamente una matriz y el bucle `foreach`, escribe un programa que presente los nombres de los días de la semana.
5. Escribe un programa que lea una matriz de números y regrese una lista de las cantidades menores a 15.
6. Escribe un programa que lea una matriz y regrese un mensaje indicando si la matriz contiene sólo valores únicos.