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

En 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
*	Multiplicación
/	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
`&&`	Y (AND)
`||`	O (OR)
`!`	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/xhtml1-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/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head><title /></head>
  <body>
    <h2>¡Éxito!</h2>
```

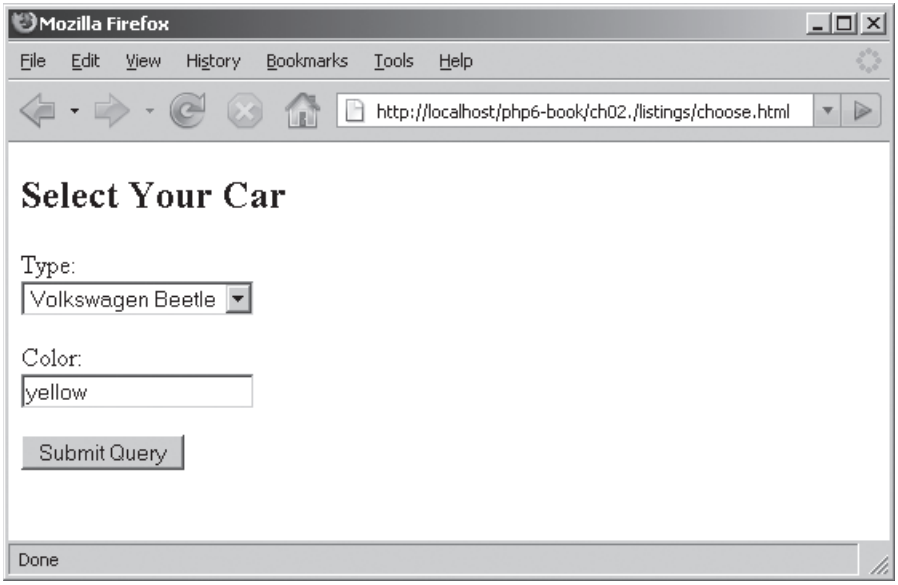

Figura 2-2 Un formulario sencillo

```
    <?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']`.

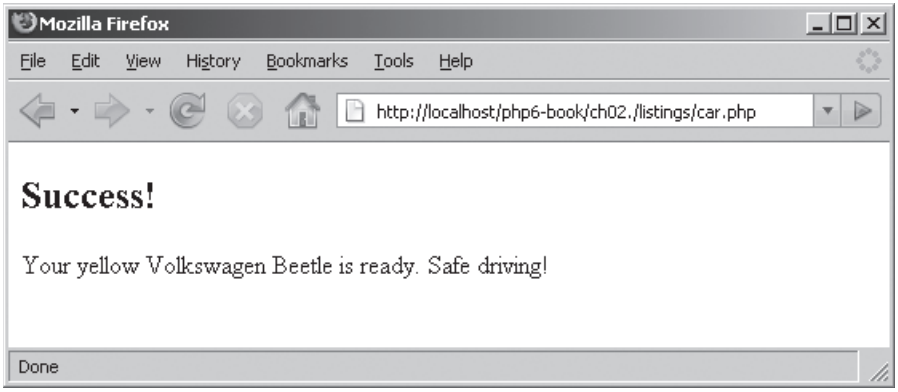

Figura 2-3 El resultado de enviar el formulario

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.