Parte I

Utilizar PHP

1

Curso acelerado de PHP

En este capítulo se realiza un rápido repaso de la sintaxis y de las estructuras de lenguaje de PHP. Si ya programa en PHP, puede utilizar este capítulo para colmar sus lagunas de conocimientos. Si tiene experiencia en programación con C, ASP u otro lenguaje de programación, este capítulo le ayudará a ponerse al día rápidamente.

En este libro aprenderá a utilizar PHP a través de una serie de ejemplos del mundo real, tomados

de nuestra experiencia en el desarrollo de sitios de comercio electrónico. Por regla general, los libros sobre programación utilizan ejemplos muy simples para explicar elementos de sintaxis. Nosotros hemos decidido no seguir esta práctica. Nos hemos dado cuenta de que lo que el lector quiere con frecuencia

es poder ver un sitio ya creado y operativo, para entender cómo se utiliza el, en lugar de otra guía de sintaxis y funciones sin mayor interés que un manual en línea.

Pruebe los ejemplos, introdúzcalos o cárguelos desde el CD-ROM, modifíquelos, divídalos en partes y aprenda a unirlas de nuevo.

En este capítulo comenzaremos por presentar un ejemplo de un formulario de pedidos de productos en línea para aprender a utilizar las variables, operadores y expresiones en PHP. También analizaremos los tipos de variables y el orden de precedencia de los operadores. Le enseñaremos a acceder a variables de formulario y a manipularlas para calcular los totales y los impuestos correspondientes en el pedido de u un cliente.

Seguidamente, desarrollaremos un ejemplo de formulario de pedido en línea utilizando nuestra secuencia de comandos de PHP para validar los datos. Examinaremos el concepto de valores Booleanos y presentaremos ejemplos de *uso* de `if`, `else`, del operador *?*: y de la instrucción `switch`. Por último, analizaremos el tema de los bucles para lo cual escribiremos código de PHP con el que generar tablas HTML repetitivas. Entre los temas clave que se analizarán están los siguientes:

- Incrustar PHP en HTML
- Agregar contenido dinámico
- Acceder a variables de formulario
- Identificadores
- Crea variables declaradas por el usuario
- Examinar tipos de variables
- Asignar valores a variables
- Declarar y utilizar constantes
- Ámbito de variables
- Operadores y su prioridad
- Evaluar expresiones
- Utilizar funciones de variables
- Toma de decisiones con `if` , `else` y `switch`
- Iteración : bucles `while` , `do` y `for`

Utilizar PHP

Para realizar los ejemplos de ese capítulo, así como los del resto del libro, necesitará disponer de acceso a un servidor Web con PHP instalado. Para sacar el máximo partido a los ejemplos y casos prácticos que se presentarán, debería ejecutarlos e intentar modificarlos. Para ello, necesitará un banco de pruebas sobre el que experimentar. Si no tiene PHP en su equipo, deberá instalarlo o pedirle a su administrador del sistema que lo haga en su lugar. En el apéndice A encontrará las instrucciones y en el CD-ROM se incluye lo necesario para instalar PHP en UNIX y Windows NT.

Creación de una aplicación: Bob’s Auto Parts

Una de las aplicaciones más comunes de cualquier lenguaje de secuencia de comandos de servidor es el procesamiento de formularios HTML. Empezaremos nuestro estudio de PHP implementando un formulario para Bob's Auto Parts, una compañía ficticia de repuestos de coches. El código utilizado para este ejemplo se incluye en el directorio correspondiente del CD.

Crear el formulario de pedidos

Por el momento, el programador de HTML de la empresa de Bob ha creado un formulario de pedido para los repuestos que vende la empresa. En la figura 1.1 se ilustra el aspecto del formulario. Resulta bastante sencillo y es muy parecido a los que se pueden encontrar en Internet. Lo primero que Bob quiere saber es qué pide el cliente, obtener el total del pedido e incluirlos impuestos que se deben pagar.

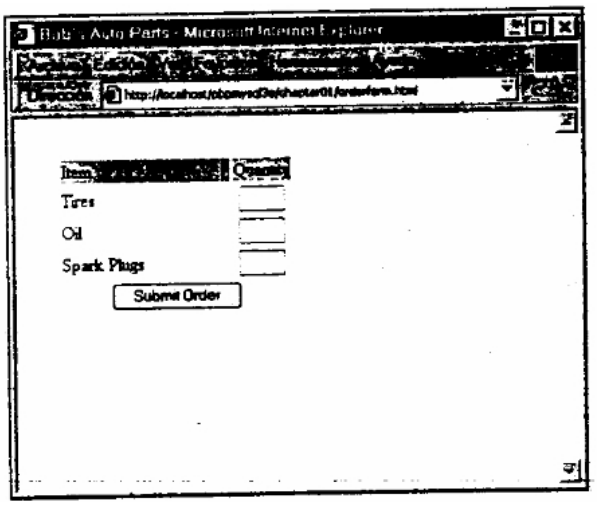

Figura 1. 1. El formulario inicial de pedido de Bob sólo registra productos y cantidades.

En el listado 1. 1 se recoge parte del código HTML asociado a este formulario.

Listado 1. 1. orderform.html. Código HTML correspondiente al formulario básico de pedido de Bob.

```
<form action="processorder.php” method=post>
<table border=0>
<tr bgcolor=#cccccc>
    <td width=150>Item</td>
    <td width=150>Quantity</td>
</tr>
```

```
<tr>
   <td>Tires</td>
   <td align=”center”><input type=”text” name= “tireqty” size”3”
maxlenght=”3”></td>
</tr>
<tr>
   <td>Oil</td>
   <td> align=”center”><input type=text”text” name=”oilqty” size=”3”
   maxlenght=”3”></td>
</tr>
<tr>
   <td>spark Plug</td>
   <td align=”center”><input type=”text” name=”sparkqty” size=”3”
   maxlenght=”3”></td>
</tr>
<tr>
   <td colspan=”2” align=”center”><input type=”submit” value=”Submi Order”></td>
</tr>
</table>
</form>
```

En primer lugar, hemos establecido la acción del formulario en el nombre de la secuencia de comandos PHP que procesará el pedido del cliente. (Escribiremos esta secuencia de comandos a continuación.) Por regla general, el valor del atributo `action` es el URL que se cargará cuando el usuario pulse el botón de envío. Los datos indicados por el usuario en el formulario se enviarán a este URL a través del método especificado en el atributo `method`, ya sea `get` (ajuntado al final del URL) o post (enviado como un paquete diferente).

En segundo lugar, debería fijarse en los nombres de los campos del formulario: `tireqty`, `oilqty` y `sparkqty`. Utilizaremos estos nombres de nuevo en nuestra secuencia de comandos de PHP. Por ello, resulta importante asignar nombres significativos a los campos de formulario que sean sencillos de recordar al escribir la secuencia de comandos de PHP.

Algunos editores HTML generan nombres de campo del tipo `campo23` de manera predeterminada. Estos campos resultan difíciles de recordar. Su vida como programador de PHP resultará más sencilla si estos nombres reflejan los datos que se introducen en el campo.

Es aconsejable adoptar un estándar de codificación para nombres de campos de manera que todos los nombres del sitio utilicen el mismo formato. De esta forma le resultará más sencillo recordar, por ejemplo, si abrevió una palabra en el nombre de un campo o si utilizó guiones bajos para representar espacios.

Procesar el formulario

Para procesar el formulario, tendremos que crear la secuencia de comandos mencionada en el atributo `action` de la etiqueta `form` llamada `processorder.php`. Abra su editor de texto y cree este archivo. Escriba el siguiente código:

```
<head>
   <title>Bob's Auto Parts - Order results </title>
```

```
</head>
<body>
    <hl>Bob’s Auto Parts</hl>
    <h2>0rder Reaults</h2>
</body>
</html>
```

Como observará, todo lo que hemos escrito hasta ahora es simple código HTML Ha llegado el momento de agregar algo de código PHP a nuestra secuencia de comandos.

Incrustar PHP en HTML

Agregue las siguientes líneas debajo del encabezado <h2>:

```
<?php
    echo '< p>Orde r processed.</p>’;
?>
```

Guarde el archivo y cárguelo en el navegador. Para ello, rellene el formulario de Bob y haga clic en el botón **Submit Order**. El resultado debería parecerse al ilustrado en la figura 1.2.

Figura 1.2. El texto pasado a la instrucción echo de PHP se refleja en el navegador.

Fíjese en cómo se incrustó el código de PHP que escribimos dentro de un archivo HTML de aspecto normal. Intente visualizar el código fuente en su navegador. Debería mostrarse la siguiente secuencia:

```
<html>
<head>
    <title>Bob"s Auto Parts - Order Re5:ulta</title>
</head>
<body>
    <hl>Bob's Auto Parts </hl>
    <h2>0rder Results</h2>
```

```
    <p>Order prosessed.</p>
</body>
</html>
```

No se ve ninguna secuencia de PHP ya que el intérprete de PHP ha recorrido la secuencia de comandos y la ha sustituido con su resultado. Por lo tanto, se puede generar código HTML limpio y visible a partir de PHP desde cualquier navegador. En otras palabras, el navegador del usuario no necesita entender PHP.

Este hecho ilustra de forma resumida el concepto de secuencias de comandos del lado del servidor. El código PHP se ha interpretado y ejecutado en el servidor Web, lo que difiere de JavaScript y de otras tecnologías de cliente que se interpretan dentro del navegador Web del equipo de un usuario.

El código de archivo se compone de cuatro elementos:

- HTML
- Etiquetas PHP
- Instrucciones PHP
- Espacios en blanco

También podemos agregar

- Comentarios

La mayor parte de las líneas de este ejemplo son sencillo código HTML.

Utilizar etiquetas PHP

El código PHP del ejemplo anterior empezaba con `<?php` y terminaba con `?>`. Es similar a las etiquetas HTML ya que todas empiezan con un símbolo menor que (<) y terminan con un símbolo mayor que (>). Estos símbolos se denominan etiquetas PHP que indican al servidor Web dónde empieza el código PHP y dónde termina. Todo el texto incluido entre las etiquetas se interpretará como PHP. El texto que se encuentre fuera de las etiquetas se procesará como HTML normal. Las etiquetas de PHP le permiten escapar de HTML.

Como veremos a continuación, existen diferentes estilos de etiquetas.

Estilo de etiquetas PHP

En PHP existen cuatro estilos diferentes de etiquetas que podemos utilizar. Los siguientes fragmentos de código son equivalentes:

- Estilo XML
  ```
  <?php echo '<P>Order processed.</P>'; ?>
  ```
  Éste es el tipo de etiqueta que utilizaremos en el libro. Se trata del estilo de etiqueta preferido que usar con PHP. El administrador del servidor no puede desactivarlo lo que garantiza su disponibilidad para todos los servidores, muy aconsejable para escribir aplicaciones que se utilicen en diferentes instalaciones. Este estilo de etiqueta se puede

utilizar con documentos XML (del: inglés Extensible Markup Language, Lenguaje de marcado extensible). Si tiene previsto utilizar XML en su sitio, debería utilizar este estilo de etiqueta.

- Estilo corto
  `<? echo '<p>Order processed.</p>'; ?>`
  Este estilo de etiqueta es el más simple y sigue el estilo de una instrucción **de** procesamiento SGML (del inglés Standard Generalized Markup Language, Lenguaje de marcado generalizado estándar). Para utilizar este tipo de etiqueta (la más corta), debe habilitar el parámetro `short_open_tag` en el archivo de configuración o compilar PHP para que utilice este tipo de etiquetas. En el apéndice A encontrará más información sobre su instalación. No se recomienda utilizar este estilo ya que, aunque está habilitado de forma predeterminada, los administradores de sistemas lo suelen desactivar porque interfiere con las declaraciones de documentos XML.

- Estilo SCRIPT
  `<script language='php'> echo '<p>Order processed.</p>';</script>`
  Este estilo de etiqueta es la más larga y le resultará familiar si conoce JavaScript o VBScript. Puede utilizarla si emplea un editor HTML que dé problemas con otros estilos de etiquetas.

- Estilo ASP
  `<? echo '< P>Order processed.</p>'; ?>`
  Este estilo de etiqueta es el que utilizan las Páginas Activas de Servidor (ASP) o ASP.NET. Se puede utilizar si ha activado el parámetro `asp_tags`. Puede recurrir a este estilo de etiquetas si utiliza un editor diseñado para ASP o ASP.NET, o si ya ha programado en ASP. De forma predeterminada aparece, deshabilitado.

Instrucciones de PHP

Para indicarle al intérprete de PHP qué hacer, se introducen instrucciones de PHP entre las etiquetas de cierre y las etiquetas de apertura. En este ejemplo, sólo utilizamos un tipo de instrucción:

```
echo  ‘<p>Order processed.</P>' ;
```

Como habrá supuesto, la función de la instrucción echo es muy sencilla: imprime (o repite) la cadena pasada en el navegador. Si examina la figura 1.2, el resultado devuelto es la cadena "`Order processed`." en el ventana del navegador.

Como observará, al final de la instrucción `echo` aparece un punto y coma. Este signo se utiliza para separar instrucciones en PHP como si se tratara de un punto al escribir frases en español. Si ha

programado antes en C o en Java, estará familiariza do con el uso del punto y coma con dicha función.

Uno de los errores de sintaxis más comunes al programar es olvidarse de introducir los puntos y comas. Sin embargo, resulta igualmente sencillo descubrir el error y corregirlo.

Espacios en blanco

Los caracteres de espaciado como las líneas nuevas (retornos del carro), los espacios y los tabuladores se conocen como espacios en blanco. Como probablemente sepa, los navegadores ignoran los espacios en blanco en HTML. Y otro tanto hace el motor de PHP. Examine los siguientes dos fragmentos de HTML:

```
<hl>Welcome to Bob's Auto Parts!</hl><p>What would y ou like to order today?</p>
```

y

```
<hl>Welcome to Bob’s
Auto Parts!</hl>
<p>What would you like
to order today?</p>
```

Estos dos fragmentos se HTML devuelven el mismo resultado al representarlos el navegador. Sin embargo, conviene utilizar espacios en blanco en HTML para mejorar la legibilidad del código. Y otro tanto se puede decir de PHP. El uso de espacios en blanco no es obligatorio pero facilita la lectura del código si colocamos cada instrucción en una línea separada. Por ejemplo:

```
echo  ‘hello’;
echo  ‘world’;
```

y

```
echo  ‘hello’; echo  ‘world’;
```

son equivalentes, pero la primera versión resulta más sencilla de leer.

Comentarios

Los comentario son exactamente eso, comentarios. Su función es la de proporcionar indicaciones para aquellos que leen el código. Los comentarios se pueden utilizar para explicar el objetivo de la secuencia de comandos, quién la escribió, por que se escribió de esa forma, cuando se modificó por última vez, etc. Por regla general, todas las secuencias de comandos de PHP incluyen comentarios, salvo las más sencillas.

Intérprete de PHP ignorará cualquier texto de un comentario. En concreto, el analizador de PHP omite los comentarios que equivalen a espacios en blanco.

PHP admite el uso de comentarios del estilo de C, C++ y secuencias de comandos del núcleo.

A continuación se recoge un comentario del estilo de C multilínea que puede aparecer al inicio de una secuencia de comandos de PHP:

```
/* Autor: Bob Smith
Modificado por última vez: 10 de Abril
Esta secuencia de comandos procesa pedidos de clientes.
 */
```

Los comentarios multilínea deben empezar y acabar con el símbolo / *. Como ocurre en C, los comentarios en línea no se pueden anidar.

También se pueden utilizar comentarios de una sola línea, bien en estilo de C++:

```
echo ‘<p>Order processed.</p>’; // Empezar a imprimir el pedido
```

o en estilo de secuencia de comandos del núcleo:

```
echo ‘<p>Order processed.</p>’; #Empezar a imprimir el pedido
```

En ambos casos, todo lo que venga después del símbolo de comentario (# o / /) se considera con un comentario hasta el final de la línea y el final de la etiqueta de PHP, según qué aparezca primero. En la siguiente línea de código, el texto que aparece por delante de la etiqueta de cierre, `esto es un comentario`, forma, parte de un comentario. El texto que aparece por detrás de la etiqueta de cierre, esto no, se considerará HTML, ya que se encuentra fuera de la etiqueta de cierre.

```
//esto es un comentario ?> esto no
```

Agregar contenido dinámico

Hasta ahora, no hemos utilizado PHP para hacer algo que no pudiésemos conseguir con HTML. La razón principal para utilizar un leguaje de secuencia de comandos de servidor es suministrar contenido dinámico a los usuarios de un sitio. Se trata de una aplicación importante porque el contenido que cambia en función de las necesidades del usuario o con el tiempo garantiza la vuelta de los visitantes. PHP nos permite realizar esta tarea de manera sencilla

Vamos a empezar por un ejemplo sencillo. Sustituya el código de PHP del archivo `processorder.php` por el siguiente código:

```
<?php
    echo '<p> Order processed at ‘;
```

```
    echo date('H:i. jS F');
    echo '</p>';
?>
```

En este fragmento, se utiliza la función `date()` integrada en PHP para indicar al cliente la fecha y la hora a la que procesar el pedido. Esta información será diferente cada vez que se ejecute la secuencia de comandos. En la figura 1.3 se ilustra el resultado de ejecutar la secuencia de comandos en una ocasión:

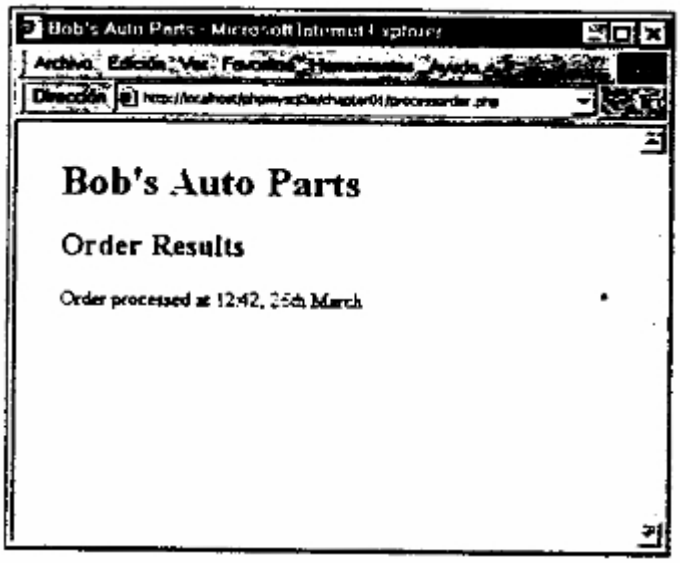

Figura 1.3. La función `date()` devuelve la cadena de fecha con formato.

Invocar funciones

Examine la llamada a la función date (). Se trata de la forma general que adoptan las llamadas de función. PHP incorpora una extensa biblioteca de funciones que puede utilizar para desarrollar aplicaciones Web La mayor parte de estas funciones pasan y devuelven datos.

Examine la siguiente llamada de función:

```
Date ('H:i, js F')
```

Fíjese en que estamos pasando una cadena (datos de texto) a la función dentro de un par de paréntesis. Esta operación se conoce como llamar al argumento o parámetro de la función. Estos argumentos son los datos de entrada utilizados por la función para devolver resultados específicos.

Utilizar la función `date()`

La función `date()` espera que el argumento que se le pase sea una cadena de formato, que represente el estilo que se desea utilizar para devolver los datos. Cada una de la s letras de la cadena representa una parte de la fecha y la hora. H es la hora en formato 24 horas, **i** son los minutos

precedidos de un cero cuando resulte necesario, `j` es el día del mes sin utilizar un cero inicial, S representa el sufijo ordinal (este caso "th" del inglés) y F es el nombre completo del mes.

(En un capítulo posterior se incluye una lista completa de los formatos que admite la función `date()`.)

Acceder a variables de formulario

El único objetivo de utilizar el formulario de pedido es recoger el pedido del cliente. Con PHP resulta muy sencillo obtener los detalles de lo que acaba de introducir el cliente, pero el método exacto depende de la versión del lenguaje que esté utilizando y de un parámetro del archivo `php.ini`.

Variables de formulario

Dentro de una secuencia de comandos de PHP, podemos acceder a cada uno de los campos del formulario como una variable de PHP cuyo nombre se relaciona con el nombre del campo del formulario. En PHP los nombres de variables se reconocen porque comienzan con el signo del dólar (`$`). Un error habitual es olvidarse de incluir este símbolo. En función de su versión de PHP y de su configuración, hay tres formas de acceder a los datos de un formulario a través de variables. Estos métodos no tienen nombres oficiales por lo que los hemos denominado e s tilo largo, intermedio y corto. En cada caso, los campos de formulario de la página enviada a la secuencia de comandos de PHP se encuentran disponibles en la secuencia de comandos. Puede acceder a los contenidos del `tireqty` de las siguientes formas:

```
$tireqty                        // estilo corto
$_POST('tireqty')               // estilo intermedio
$HTTP_POST_VARS ('tireqty')     // estilo largo
```

En este ejemplo, y a lo l argo del libro, se utiliza el estilo intermedio (es decir, `$_POST ['tireqty']`) para hacer referencia a las variables de formulario pero puede crear versiones cortas de las variables para facilitar su uso. (Se trata del enfoque recomendado de s de la versión 4.2.0 de PHP.)

En su código, puede utilizar otro enfoque, pero para ello debería conocer las diferentes opciones. En resumen:

- El estilo corto (`$tireqty`) es práctico, pero requiere activar el parámetro de configuración register_globals. (El valor predeterminado varía de una versión a otra de PHP.) En todas las versiones desde la 4.2.0 se desactiva de forma predeterminada. Antes, aparecía habilitado y casi todos los programadores lo utilizaban. Este cambio ha provocado gran confusión desde su implementación. En este estilo resulta sencillo cometer errores que podrían convertir a su código en inseguro, por lo que no es una opción recomendada.

- El estilo intermedio (`$_POST ['tireqty']`) se ha convertido en el enfoque recomendado. Es bastante práctico, pero hizo su aparición en la versión PHP 4 .1.0, por lo que no funcionará en las versiones antiguas.

- El estilo largo (`$HTTP_POST_VARS ['tireqty']`) es el que incluye más detalles. Sin embargo, tenga en cuenta que se considera obsoleto y que es probable que con el tiempo acabe desapareciendo. Solía ser el estilo más portátil pero se puede desactivar a través de la directiva de configuración `register_long_arrays`, lo que mejora el rendimiento.

En el estilo corto, los nombres de las variables utilizadas en la secuencia de comandos son iguales a los nombres de los campos del formulario HTML. No es necesario declarar las variables en la secuencia de comandos o adoptar ninguna medida para crear dichas variables. Se pasan en la secuencia de comandos básicamente de la misma forma que se pasan argumentos a una función. Si ha escogido este estilo, puede utilizar una variable como `$tireqty`. El campo `tireqty` del formulario crea la variable `$tireqty` en la secuencia de comandos de procesamiento. Resulta práctico disponer de este tipo de acceso a las variables, pero antes de activar el parámetro `register_globals` conviene conocer las razones que han llevado al equipo de desarrollo de PHP a dejarlo desactivado.

Como hemos dicho, este tipo de acceso a las variables resulta práctico, pero es proclive a cometer errores de programación que podrían comprometer la seguridad de las secuencias de comandos. Si las variables de formulario se convierten automáticamente en variables globales, no existirá una diferencia clara entre las variables creadas por nosotros y las variables no fiables procedentes d i rectamente del usuario.

Si no asignamos un valor inicial a nuestras variables, los usuarios de las secuencias de comandos pueden pasar variables y valores en forma de variables de formulario que se mezclarán con las nuestras. Si se opta por utilizar el estilo corto para acceder a las variables, es aconsejable asignar un valor inicial a las variables.

El estilo intermedio implica la recuperación de variables de formulario de una de las matrices `$_POST`, `$_GET` y `$_REQUEST`. Una de las matrices `$_GET` o `$_POST` contendrá los detalles de todas las variables de formulario: la matriz utilizada dependerá del método utilizado para enviar el formulario, POST o GET, respectivamente. Además, todos los datos enviados a través de los métodos POST y GET estarán disponibles a través de `$_REQUEST`.

Si el formulario se envía por el método POST, los datos introducidos en el cuadro `tireqty` se almacenarán en `$_POST ['tireqty']`. Si el formulario se envía utilizando GET, los datos se almacenarán en `$_GET ['tireqty']`. En cualquiera de los dos casos, los datos estarán disponibles en `$_REQUEST ['tireqty']`.

Estas matrices son algunas de las denominadas matrices superglobales, a las que volveremos al examinar el ámbito de las variables.

Si está utilizando una versión antigua de PHP, puede que no disponga de acceso a `$_POST` o `$_GET`. En las versiones anteriores a la 4.1.0, esta información se almacenaba en las matrices `$HTTP_POST_ VARS` y `$HTTP_GET_ VARS`. Este estilo es el que llamaremos estilo largo y se puede utilizar con versiones antiguas y nuevas de PHP, pero ha sido considerado como obsoleto por lo

que es probable que no funcione en las versiones futuras. En este estilo no existe un equivalente de $_REQUEST.

Si utiliza el estilo largo, puede acceder a la respuesta del usuario a través de $HTTP_POST_ VARS ['tireqty'] o $HTTP _GET_ VARS ['tireqty'].

Los ejemplos utilizados en este libro se han probado con la versión 5.0 de PHP, por lo que es posible que a veces resulten incompatibles con versiones anteriores a la 4.1.0. Por lo tanto, le recomendamos que utilice la versión actual.

Veamos otro ejemplo. Como los nombres de variable resultan un tanto pesados en el estilo largo e intermedio, y se basan en un tipo de variables conocidas como matrices, que no se analizarán de lleno hasta un capítulo posterior, comenzaremos por crear copias que resulten sencillas de utilizar.

Para copiar el valor de una variable en otra, puede utilizar el operador de asignación, que en PHP es el signo igual (=). La siguiente línea de código crea una nueva variable denominada $tireqty y copia los contenidos de $HTTP_POST_VARS ['tireqty'] dentro de la nueva variable:

```
$tireqty = $HTTP_POST_VARS ['tireqty'];
```

Coloque el siguiente bloque de código al inicio de la secuencia de comandos de procesamiento. El resto de secuencias de comandos de este libro que procesen datos desde un formulario contendrás un bloque similar al inicio. Como no se generará ningún resultado, no importa si se coloca por encima o por debajo de la etiqueta <html> ni de otras etiquetas de HTML con las que se inicie la página. En nuestro caso, solemos colocar este bloque al inicio de la secuencia de comandos para asegurarnos de que resulta fácil de encontrar.

```
<?php
    //cree nombres de variables cortos
    $tireqty = $HTTP_POST_VARS ['tireqty'];
    $oilqty = $HTTP_POST_VARS ['oilqty'];
    $sparkqty = $HTTP_POST_VARS ['sparkqty']
?>
```

El código crea tres nuevas variables $tireqty, &oilqty, $sparkqty, y las configura para que contengan los datos enviados a través del método POST desde el formulario.

Para que la secuencia de comandos haga algo visible, agregue las siguientes líneas a la parte final de la secuencia de comandos PHP:

```
echo '<p>Your order is as follows: </p>';
echo $tireqty.' Tires <br />';
echo $oilqty.' Bottles of oil<br />';
echo $sparkqty.' Spark plugs<br />';
```

Por el momento, no hemos comprobado el contenido de las variables para asegurarnos de que se han introducido datos correctos en los campos del formulario. Pruebe a introducir datos incorrectos para ver qué sucede. Una vez leído el resto del capítulo, puede que le interese añadir algún método de validación de datos a esta secuencia de comandos.

Si carga ahora este archivo en el navegador, el resultado de la secuencia de comandos se parecerá al ilustrado en la figura 1.4. Los valores reales se mostrarán en función de lo que escriba en el formulario.

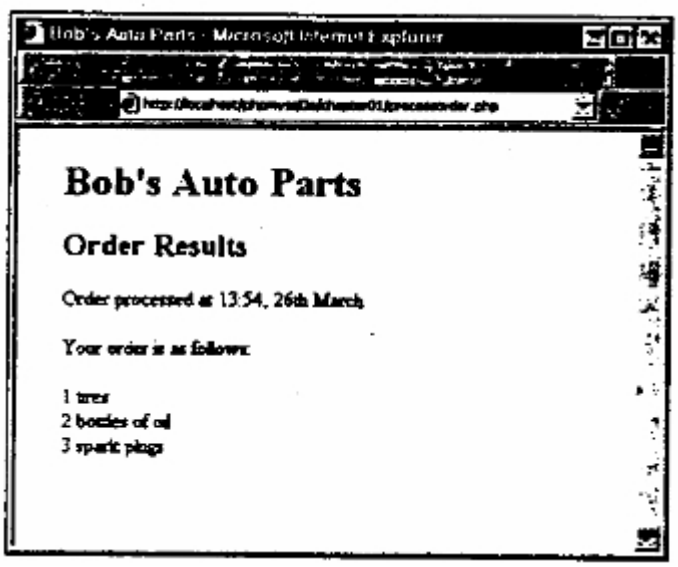

Figura 1.4. Las variables de formulario introducidas por el usuario sepultan de fácil acceso en `processor.php`.

En las siguientes secciones se comentan un par de elementos interesantes sobre este ejemplo

Concatenar cadenas

En la secuencia de comandos, utilizamos `echo` para imprimir el valor introducido por el usuario en cada uno de los campos de formulario, seguido por texto explicativo. Si examina de cerca las instrucciones `echo`, verá que el nombre de la variable y que el texto que sigue están separados por un punto (.), como se puede ver a continuación:

```
echo $tireqty. 'tires <br />';
```

Éste es el operador de concatenación de cadenas y se utiliza para unir cadenas (secciones de texto). Lo utilizara con frecuencia con `echo` para mostrar información en pantalla. Se suele recurrir a él para no tener que escribir varios comandos `echo`. Para las variables que no sean matrices, puede encerrarlas entre comillas dobles para su representación en el navegador. (El estudio de las matrices se aborda en un capítulo posterior.) Por ejemplo:

```
echo "$tireqty tires <br />";
```

Esta secuencia es equivalente a la primera. Los dos formatos son válidos y la selección de uno u otro es una cuestión personal. El proceso de sustituir una variable con sus contenidos dentro de una cadena se denomina interpolación.

La interpolación es exclusiva de las cadenas entre comillas dobles. No se puede añadir nombres de variables dentro de una cadena entre comillas simples de esta forma. Si se ejecuta la siguiente línea de código:

```
echo ‘$tireqty tires<br />’;
```

Enviará la secuencia "$tireqty tires<br / >" al navegador. Si se utilizan comillas dobles, el nombre de la variable se sustituirá por su valor. Si se utilizan comillas simples se enviará el nombre de la variable, o cualquier otro texto, sin alterar.

Variables y literales

Las variables y las cadenas concatenadas en cada una de las instrucciones `echo` son cosas diferentes. Las variables son un símbolo para sustituir a los datos. Las cadenas son datos en sí mismas. Cuando se utiliza un segmento de datos como tal en un programa como éste, se designa como literal para distinguirlo de las variables. `$tireqty` es una variable, un símbolo que representa los datos escritos por el cliente. Por el contrario, `' tires<br / >'` es un literal. Se puede tomar por su valor literal. Bueno, casi. ¿Recuerda el segundo ejemplo anterior? PHP sustituye el nombre de la variable `$tireqty` en la cadena por el valor almacenado en la variable.

Recuerde que existen dos tipos de cadenas en PHP, unas encerradas entre comillas dobles y otras entre comillas simples. PHP intentará evaluar las cadenas con comillas dobles, lo que dará como resultado el comportamiento visto anteriormente. Las cadenas con comillas simples se tratarán como verdaderos literales.

Recientemente ha aparecido una tercera forma de especificar cadenas. La sintaxis heredoc (<<<), que a los usuarios de Perl le resultará familiarizar se añadió a PHP4. Esta sintaxis permite especificar extensas cadenas de forma elegante, mediante un marcador final que se utilizará para finalizar la cadena. El siguiente ejemplo crea una cadena de tres líneas y la reproduce:

```
echo <<<theEnd
         línea 1
         línea 2
         línea 3
theEnd
```

El elemento theEnd es totalmente arbitrario. Simplemente no debe aparecer en el texto. Para cerrar una cadena heredoc debe incluir un elemento de cierre al comienzo de la línea. Las cadenas heredoc se interpolan como las cadenas entre comillas dobles.

Identificadores

Los identificadores son nombres de variables. (Los nombres de funciones y clases son también identificadores. Los nombres y las clases se examinarán en capítulos posteriores.) Los identificadores siguen algunas reglas sencillas:

- Los identificadores pueden tener cualquier longitud y pueden incluir letras, números y guiones bajos.
- Los identificadores no pueden comenzar por un número.

- En PHP, los identificadores discriminan entre mayúsculas y minúsculas. `$tireqty` no es lo mismo que `$TireQty`. Un error común en programación consiste en intentar utilizarlos indistintamente. Los nombres de las funciones son una excepción a esta regla (el uso de mayúsculas y minúsculas es indiferente).
- Una variable puede tener el mismo nombre que una función. Este hecho puede dar lugar a confusiones por lo que conviene evitarlos. Así mismo, no se pueden crear dos funciones con el mismo nombre.

Crear variables declaradas por el usuario

Puede declarar y utilizar sus propias variables además - de las variables que se pasan desde el formulario HTML.

Uno de los rasgos de PHP es que no requiere declarar variables antes de utilizarlas. Las variables se crean al asignarles un valor (véase la siguiente sección para conocer los detalles).

Asignar valores a variables

Para asignar valores a las variables se utiliza el operador de asignación (=) como hicimos al copiar el valor de una variable en otra. En el sitio de Bob, queremos obtener el número total de artículos pedidos y la cantidad total que debe abonar. Podemos crear dos variables para almacenar restos valores. En primer lugar, inicializamos cada una de estas variables en cero.

Agregue las siguientes líneas al final de la secuencia de comandos de PHP:

```
$totalqty = 0;
$totalamount = 0.00;
```

Cada una de estas líneas crea una variable y le asigna un valor literal. También puede asignar valores de variables a las variables, por ejemplo:

```
$totalqty = 0;
$totalamaount = 0 $totalqty;
```

Tipos de variables

Un tipo de variable hace referencia al tipo de datos que se almacenan en ella. PHP cuenta con un completo conjunto de tipos de datos. En los distintos tipos de datos se pueden almacenar diferentes datos.

Tipos de datos de PHP

PHP admite los siguientes tipos de datos

- **Entero:** Utilizado para números enteros
- **Flotante (o Doble):** Utilizado para números reales
- **Cadena:** Utilizado para cadena de caracteres
- **Booleano**: Utilizado para valores verdaderos o falsos
- **Matriz**: Utilizado para almacenar conjuntos de datos de mismo tipo (véase un capítulo posterior) 0
- **Objeto:** Utilizado para almacenar instancias de clases (véase un capitulo posterior)

También existen dos tipos especiales: NULL y de recurso. Las variables a las que no se le ha asignado un valor, no están definidas o se les ha asignado el valor NULL son de tipo NULL.

Algunas funciones incorporadas (como las funciones de base de datos) devuelven variables con

tipo de recursos. Representan recursos externos (como conexiones de base de datos). Es muy poco probable que necesite manipular directamente este tipo de variables, pero las funciones les suelen devolver y deben pasarse como parámetros a otras funciones.

Control de tipos

PHP es un lenguaje con un control de tipos muy débil. En la mayoría de los lenguajes, las variables sólo pueden contener un tipo de datos y dicho tipo de datos debe declararse antes de poder utilizar la variable, como ocurre en C. En PHP, el tipo de variable viene determinado por el valor que se asigne.

Por ejemplo, al crear `$totalqty` y `$totalamount`, deben determinarse sus tipos iniciales, de la siguiente forma:

```
$totalqty = 0;
$totalamaount = 0.00;
```

Como hemos asignado 0, un valor entero, a `$totalqty,` se tratará de una variable de tipo entero. De manera similar, `$totalamount` es de tipo doble.

Pero, por extraño que pueda parecer, podemos agregar una línea a nuestra secuencia de comandos de la siguiente forma:

```
$totalamount = 'Hello';
```

La variable `$totalamount` será entonces de tipo cadena. PHP cambia el tipo de la variable en función del valor almacenado en ella en cualquier momento dado.

Esta capacidad para cambiar los tipos de manera transparente al instante puede resulta extremadamente útil. Recuerde que PHP sabe “automáticamente” qué tipo de datos se añade a una variable y devolverá los datos con el mismo tipo tras recuperarlos de la misma.

Convertir tipos

Puede simular que un tipo de variable o valor sea de un tipo diferente utilizando una conversión de tipo. Este recurso funciona de la misma forma que en C. Basta con colocar el tipo temporal entre

corchetes delante de la variable que se desea convertir. Por ejemplo, podríamos haber declarado las dos variables anteriores utilizando una conversión.

```
$totalqty = 0;
$totalamount = (double) $totalqty;
```

La segunda línea indica “toma el valor almacenado en `$totalqty`, interprétalo como doble y almacénalo como `$totalamount` ". La variable `$totalamount` será de tipo doble. La variable de conversión no cambia de tipo por lo que `$totalqty` seguirá siendo de tipo entero.

Variables de tipo variable

PHP proporciona otro tipo de variable: la variable de tipo variable. Las variables de tipo variable permiten cambiar el nombre de una variable dinámicamente.

(Como puede observar, PHP proporciona una gran libertad en este sentido: todos los lenguajes permiten cambiar el valor de la variable, pero no hay muchos que permitan cambiar el tipo d e variable y menos aún que permitan cambiar su nombre.) Su funcionamiento consiste en utilizar el valor de una variable como nombre de otra. Por ejemplo, podemos establecer

```
$varname = ‘tireqty’;
```

A continuación podemos utilizar `$varname` en lugar de `$tireqty`. Por ejemplo, podemos establecer el valor de `$tireqty`:

```
$$varname = 5;
```

Esto equivale a

```
$tireqty = 5;
```

Puede que parezca un poco confuso, pero no se preocupe porque volveremos sobre su uso posteriormente. En lugar de tener que enumerar y utilizar cada variable de formulario por separado, podemos utilizar un bucle y una variable para procesarlas todas automáticamente. En la sección dedicada a los bucles `for` se incluye un ejemplo que ilustra este hecho.

Declarar y utilizar constantes

Como vimos anteriormente, podemos cambiar el valor almacenado en una variable. También podemos declarar constantes. Una constante almacena un valor como una variable con la diferencia de que se establece una vez y no se puede cambiar en ningún otro punto de la secuencia de comandos.

En nuestra aplicación de ejemplo, podríamos almacenar los precios de los artículos de venta en forma de constantes. Para definir constantes puede utilizar la función define:

```
define (‘TIREPRICE’, 100);
define (‘OILPRICE' , 10);
define (‘SPARKPRICE', 4 );
```

Agregue estas líneas de código a la secuencia de comandos. Ahora dispone de tres constantes que puede utilizar para calcular el total del pedido de un cliente.

Los nombres de las constantes utilizan siempre mayúsculas. Se trata de una convención tomada de C que facilita su distinción de las variables. Esta convención no es obligatoria pero contribuye a que la lectura y el mantenimiento del código resulten más sencillos.

Tenemos tres constantes que se pueden utilizar para calcular el total del pedido del cliente.

Una diferencia importante entre las constantes y las variables es que cuando hacemos referencia a una constante, no lleva antepuesto el símbolo del dinero ($). Si desea utilizar el valor de una constante, utilice únicamente su nombre. Por ejemplo, para utilizar una de las constantes que acabamos de crear, podemos escribir:

```
echo TIREPRICE;
```

Además de las constantes definidas, PHP establece un gran número de constantes propias. Una forma sencilla de verlas consiste en ejecutar el `comandophpinfo()`:

```
phpinfo();
```

Este comando devuelve una lista de variables y constantes predefinidas de PHP, además de otra información útil. Iremos comentándola a medida que vayamos avanzando. Otra diferencia entre variables y constantes es que éstas sólo pueden almacenar datos de tipo Booleano, entero, flotante o de cadena. Estos tipos se conocen como valores escalares.

Ámbito de variables

El término ámbito hace referencia a los lugares dentro de las secuencias de, comandos en los que resulta visible una variable dada. A continuación se describen las seis reglas que se aplican a los ámbitos de PHP:

- Las variables superglobales incorporadas resultan siempre visibles dentro de una secuencia de comandos

- Las constantes, una vez declaradas, siempre resultan visibles de forma global, y se pueden utilizar dentro y fuera de una función.

- Las variables globales declaradas en una secuencia de comandos resultan visibles a lo largo de la secuencia de comandos pero no dentro de las funciones.

- Las variables utilizadas dentro de funciones que se declaran como globales hacen referencia a la variable global del mismo nombre.

- Las variables creadas dentro de funciones y declaradas como estáticas son invisibles desde el exterior de la función pero conservan su valor entre las ejecuciones de ésta, como veremos en un capítulo posterior.

- Las variables creadas dentro de funciones tienen restringido su ámbito a la función y dejan de existir cuando ésta desaparece.

A partir de la versión 4.2 de PHP, las matrices `$_GET` y `$_POST` así como otras variables especiales llevan asignadas sus propias reglas de ámbito. Estas se conocen como superglobables y se pueden ver en todas partes, tanto dentro como fuera de las funciones.

A continuación se recoge la lista completa de variables globales:

- `$GLOBALS`, una matriz con todas las variables globales. (Al igual que la palabra clave global, nos permite acceder a las variables globales dentro de una función, por ejemplo como `$GLOBALS ['myvariable']`.)

- `$ _ SERVER`, una matriz con las variables de entorno del servidor

- `$_GET`, una matriz con las variables pasadas a la secuencia de comandos a través del método `GET`

- `$_POST`, una matriz con las variables pasadas a la secuencia de comandos a través del método `POST`

- `$_COOKIE`, una matriz con las variables de cookies

- `$_FILES`, una matriz con las variables relacionadas con las cargas de archivos

- `$_ENV`, una matriz de variables de entorno

- `$_REQUEST`, una matriz con todas las variables de entrada de usuario, incluido los contenidos de entrada de `$_GET`, `$_POST` y `$_COOKIE`

- `$_SESSION`, una matriz con las variables de sesión

Volveremos sobre cada uno de estos tipos a medida que vayamos avanzando. Analizaremos el ámbito de las variables al estudiar las funciones. Por el momento. Todas las variables que utilicemos serán globales de manera predeterminada.

Operadores aritméticos

Los operadores aritméticos son bastantes claros: se trata de los operadores matemáticos de uso más común. En la tabla 1.1 se recogen los operadores aritméticos.

Tabla 1.1. Operadores aritméticos de PHP.

Operador	Nombre	Ejemplo
+	Suma	$a + $b
-	Resta	$a - $b
*	Multiplicación	$a * $b
/	División	$a / $b
%	Módulo	$a % $b

Con todos estos operadores podemos guardar el resultado de la operación. Por ejemplo:

```
$result = $a + $b;
```

La suma y la resta funcionan como se esperaría. Estas operaciones suman o restan, respectivamente, los valores almacenados en las variables $a y $b. También puede utilizar el símbolo de resta (-) como operador unitario (es decir, un operador que toma un argumento u operando) para indicar valores negativos; por ejemplo:

```
$a = -1;
```

La multiplicación y la división también funcionan de la forma esperada. Tenga en cuenta que se usa el asterisco como operador de multiplicación en lugar de símbolo habitual de multiplicación y la barra inclinada como símbolo de la división. El operador de módulo devuelve el resto de la división de la variable $a por la variable $b. Considere el siguiente fragmento de código:

```
$a = 27;
$b = 10;
$result = $a/$b;
```

El valor almacenado en la variable $_result es el resto de dividir 27 por 10, es decir, 7. Tenga en cuenta que los operadores aritméticos se suelen aplicar a las variables enteras o dobles. Si se aplican a cadenas, PHP intentará convertir la cadena en un número. Si contiene una "e " o una " E ", la convertirá en una variable doble, de lo contrario la convertirá en un entero. PHP busca números al principio de la cadena y los utiliza como valores (si no hubiera ninguno, el valor de la cadena sería cero)

La variable $result contendrá la cadena “Bob’s Auto Parts”.

Operadores de asignación

Ya hemos visto el operador de asignación básico =. A este operador se hará siempre referencia como operador de asignación y se lee como "establecer en". Por ejemplo:

```
$totalqty = 0;
```

Esta secuencia se lee como “`$totalqty` se establece en cero”. Ya explicaremos por qué al hablar de los operadores de comparación en una sección posterior.

Devolver calores de asignación

El uso del operador de asignación devuelve siempre un valor general similar a otros operadores. Si escribe

```
$a +$b
```

El valor de la expresión es el resultado de agregar las variables `$a` y `$b` .
De manera análoga, puede escribir:

```
$a = 0;
```

El valor de toda expresión es cero.
Esta secuencia permite realizar operaciones como:

```
$b = 6 + ($a=5);
```

Esta secuencia establece el valor de la variable `$b` en 11. Por regla general, el valor de toda la instrucción de asignación es el valor que se asigna en el operando de la izquierda. Al calcular el valor de una expresión, se pueden utilizar los paréntesis para asignar prioridad a una subexpresión como hemos hecho en el ejemplo. Su funcionamiento es el mismo que en matemáticas.

Combinar operadores de asignación

Además de la asignación simple, existe un conjunto de operadores de asignación combinados. Se trata de formas abreviadas de realizar otra operación sobre una variable y de asignarle el resultado. Por ejemplo:

```
$a += 5;
```

Esta secuencia equivale a escribir:

```
$a = $a + 5;
```

Existen operadores de asignación combinados para cada uno de los operadores aritméticos así como para el operador de concatenación de cadenas. En la tabla 1.2 se recoge un resumen de todos los operadores de asignación combinados y su efecto.

Tabla 1.2. Operadores deasignación combinados de PHP.

Operador	Uso	Equivalente
+=	$a += $b	$a = $a + $b
-=	$a -= $b	$a = $a - $b
*=	$a *= $b	$a = $a * $b
/=	$a /= $b	$a = $a / $b
%=	$a %= $b	$a = $a % $b
.=	$a .= $b	$a = $a . $b

Incremento y decremento previo y posterior

Los operadores de incremento (++) y decremento (--) previo y posterior son similares a los operadores += y -=, pero con un par de variaciones.

Los operadores de incremento presentan dos efectos: incrementan un valor y lo asignan. Considere la siguiente secuencia:

```
$a=4;
echo ++$a;
```

La segunda línea utiliza el operador de incremento previo, denominado así porque ++ aparece delante de $a. Este operador tiene dos efectos: en primer lugar incrementa $a en una unidad y en segundo lugar devuelve el valor incrementado. En este caso, $a se incrementa en 5 y se devuelve e imprime dicho valor. El resultado de toda la expresión es 5. (Fíjese en que el valor que se almacena en realidad en $a ha variado: no estamos devolviendo simplemente $a + 1.)

Sin embargo, si colocamos ++ después de $a, utilizamos el operador de incremento posterior. Su efecto es diferente al anterior. Considere la siguiente secuencia:

```
$a=4;
echo $a++;
```

En este caso, los efectos se invierten. En primer lugar, se devuelve e imprime el valor de $a y en segundo lugar se incrementa. El resultado de toda la expresión es 4. Este es el valor que se imprimirá.

Sin embargo, el valor de $a tras ejecutar esta instrucción será 5, Como probablemente habrá adivinado, el comportamiento es similar para el operador --, con la diferencia de que el valor de $a se reduce en lugar de incrementarse.

Referencias

Una interesante utilidad es el operador de referencia, &, que se utiliza en combinación con el operador de asignación. Por regla general cuando una variable se asigna a otra, se realiza una copia de la primera y se almacena en memoria. Por ejemplo:

```
$a = 5;
$b = $a;
```

Estas líneas de código realizan una segunda copia del valor almacenado en `$a`
Y lo guardan en `$b`. Si cambiamos el valor de `$a` en un momento posterior, `$b` no variará:

```
a = 7; // $b seguirá siendo 5
```

Para evitar que se haga copia y que se guarde en memoria puede utilizar el operador de referencia, `&.` Por ejemplo

```
$a = 5;

$b = &$a;
$a = 7; // $a y $b son ahora 7 las dos
```

Las referencias pueden resultar un tanto complejas. Recuerde que una referencia es como un alias, no como un puntero.
Tanto `$a` como `$b` apuntan a la misma información. Puede cambiarlo si anula la configuración de una de ellas:

```
Unset ($a);
```

Al realizar esta operación no se modifica el valor de `$b (7)` pero anula el vínculo entre `$a` y el valor 7 almacenado en memoria.

Operadores de comparación

Los operadores de comparación se utilizan para comparar dos valores. Las expresiones que utilizan estos operadores devuelven el valor lógico `true` o el valor lógico `false` en función del resultado de la comparación.

El operador iguales

El operador de comparación iguales, == (dos signos iguales), permite determinar si dos valores son iguales. Por ejemplo, podemos utilizar la expresión

```
$a == $b
```

Para determinar si los valores almacenados en $ay en $b son iguales. El resultado devuelto por esta expresión será `true` si son iguales o `false` si no lo son.

Resulta sencillo confundir este operador de asignación (=). Esta confusión no generará un error pero impedirá que se obtengan los valores deseados. En general, los valores distintos a cero se evalúan como `true` y los valores iguales a cero se evalúan como `false`. Suponga que hemos inicializado dos variables de la siguiente forma:

```
$a = 5;
$b = 7;
```

Si prueba `$a = $b`, el resultado será `true`. ¿Por qué? El valor de `$a = $b` es el valor asignado a la parte izquierda de la expresión, que en este caso es 7. Se trata de un valor distinto a cero, por lo que la expresión se evalúa como `true`. Si su intención es probar `$a = = $b`, que devuelve `false`, habrá introducido un error de lógica en su código que puede resultar extremadamente difícil de detectar. Compruebe siempre el uso de estos dos operadores y verifique si ha utilizado el que tenía previsto.

Se trata de un error muy sencillo de cometer y es muy probable que caiga en él muchas veces a lo largo de su carrera como programador.

Otros operadores de comparación

PHP admite otros operadores de comparación, que se recogen en la tabla 1.3. Uno de los más destacados es el nuevo operador de identidad, ===, introducido en PHP, que devuelve `true` sólo si los dos operadores son iguales y del mismo tipo.

Tabla 1.3. Operadores de comparación en PHP.

Operado	Nombre	Uso
==	Igual	$a == $b
===	Idéntico	$a == $b
!=	Distinto	$a != $b
!==	Distinto	$a !== $b
<>	Distinto	$a <> $b
<	Menor que	$a < $b
>	Mayor que	$a > $b
<=	Menor o igual que	$a <= $b
>=	Mayor o igual que	$a >= $b

Operadores lógicos

Los operadores lógicos se utilizan para combinar los resultados de condiciones lógicas. Por ejemplo, puede que nos interese que el valor de una variable, `$a`, se encuentre entre 0 y 100. Para

ello tendríamos que probar las condiciones `$a >= 0` y `$a <= 100`, utilizando el operador AND, como se indica a continuación:

```
$a >= 0 && $a <=100
```

PHP admite el uso de los operadores lógicos AND, OR, XOR(o exclusivo) y NOT. En la tabla 1.4 se resumen los operadores lógicos y su uso

Operador	Nombre	Uso	Resultado
!	NOT	!$b	Devuelve true si $b es false y viceversa
&&	AND	$a && $b	Devuelve true si $a y $b son true; de lo contrario devuelve false
||	OR	$a || $b	Devuelve true si $a y $b o ambas son true; de lo contrario devuelve false
and	AND	$a and $b	Igual que &&, pero con prioridad más baja
or	OR	$a or $b	Igual que ||, pero con prioridad más baja

Los operadores and y or tienen una prioridad inferior a la de los operadores && y ||. En una sección posterior se analizará el tema de la prioridad.

Operadores bit a bit

Los operadores bit a bit permiten tratar un entero como una serie de bits utilizados para representarlos. Estos operadores no se utilizan demasiado en PHP. En la tabla 1.5 se recoge un resumen de los operadores bit a bit.

Tabla 1.5. Operadores bit a bit de PHP.

Operador	Nombre	Uso	Resultado
&	AND bit a bit	$a & $b	Los bits asignados en $ay $b se establecen en el resultado
|	OR bit a bit	$a | $b	Los bits asignados en $a o $b se establecen en el resultado y viceversa
-	NOT bit a bit	-$a	Los bits asignados en $a no se establecen en el resultado y viceversa
^	XOR bit a bit	$a ^ $b	Los bits asignados en $a o $b pero no en ambos se establecen en el resultado
<<	Desplazamiento a la izquierda	$a << $b	Mueve $a a la izquierda en función de los bits de $b
>>	DEsplamiento a la derecha	$a >> $b	Mueve $a a la derecha en función de los bits de $b

Otros operadores

Además de los operadores vistos hasta ahora, existen otros. El operador coma (,) se utiliza para separar argumentos de función y otros elementos de lista .suele utilizar de manera ocasional.

Los operadores `new` y `->` se utilizan para crear instancias de una clase y para acceder a los miembros de clase, respectivamente. Estos operadores se analizarán detalladamente en un capítulo posterior.

Existen otros tres operadores que se examinan brevemente a continuación.

Operador ternario

Este operador, ?, funciona de la misma forma que en C. Adopta la siguiente forma;

condición? valor si true : valor si false

El operador temario es similar a la versión de la expresión de una instrucción, `if - else` , que *se* analizará más adelante en este capítulo .Vemos un sencillo ejemplo:

```
($grade> 50 ? 'Aprobado' : ‘supenso');
```

Esta expresión evalúa la nota del estudiante como ‘Aprobado’ o ‘Suspenso’.

Operador de supresión de error

El operador de supresión de error, @, se puede utilizar por delante de cualquier expresión, es decir, de todos aquellos elementos que generen o tengan un valor. Por ejemplo:

```
$a * @ (57/0);
```

Sin el operador @, esta línea generará un aviso de división por cero (inténtelo). Si se incluye el operador, se suprimirá el error.

Si tiene previsto suprimir advertencias de esta forma, debería escribir código de control de errores que compruebe cuándo tiene lugar una advertencia. Si ha configurado PHP con la función `track_errors` activada, el mensaje de error se almacenará en la variable global `$php_errormsg`.

Operador de ejecución

En realidad el operador de ejecución está formado por un par de operadores: dos acentos graves (``). Este símbolo es diferente a la comilla simple (en el teclado se sitúa en la tecla con el símbolo de diéresis).

PHP intentará ejecutar todo lo que se incluya entre estos símbolos como un comando en la línea de comandos del servidor. El valor de la expresión es el resultado del comando, Por ejemplo, en sistemas operativos del tipo UNIX, puede utilizar:

```
&out = dir c:;
echo ‘<pre>’ $out. ‘</pre>’;
```

Cualquiera de estas versiones devolverá un listado de directorio y lo almacenará en &out. A continuación se puede imprimir en el navegador o manipular de otra forma. Existen otras formas de ejecutar comandos en el servidor, como veremos en capítulo posterior.

Operador de matriz

Existen diferentes operadores de matriz. Los operadores de elemento de matriz (`[]`) le permiten acceder a los elementos de una matriz. También puede utilizar el operador => indeterminados casos. Los analizaremos en uno de los siguientes capítulos, aunque se los mostramos a continuación.

Tabla 1.6 Operadores de matriz de PHP.

Operador	Nombre	Uso	Resultado
`+`	Unión	`$a + $b`	Devuelve una matriz que contiene todo lo que incluido en `$a` y `$b`
`==`	Igualdad	`$a == $b`	Devuelve `true` si `$a` y `$b` tienen los mismos elementos
`===`	Identidad	`$a === $b`	Devuelve `true` si `$a` y `$b` tienen los mismos elementos en el mismo orden
`!`	Desigualdad	`$a != $b`	Devuelve `true` si `$a` y `$b` no son iguales
`<>`	Desigualdad	`$a <> $b`	Devuelve `true` si `$a` y `$b` no son iguales
`!==`	No idéntico	`$a !== $b`	Devuelve `true` si `$a` y `$b` no son idénticos

Observará que los operadores de matriz descritos en la tabla cuentan con operadores equivalentes que funcionan con variables escalares. Siempre que recuerde que + realiza sumas en tipos variables y sirve para unir matrices, aunque no le interese la aritmética de la operación, ambos comportamientos tendrán sentido. No se pueden comparar matrices con tipos escalares.

El operador de tipo

Existe un operador de tipo: `instanceof`, que se utiliza en la programación orientada a objetos, tema que analizaremos en un capítulo posterior.

El operador `instanceof` le permite comprobar si un objeto es una instancia una clase concreta, como se indica en el siguiente ejemplo:

```
class sampleClass{};
```

```
SmyObject = news sampleClass();
if ($my0bject instance of sampleClass)
    echo “myObject is an instance of sampleClass”;
```

Utilizar operadores: calcular los totales de los formularios

Ahora que ya sabemos cómo utilizar operadores de PHP, podemos calcular los totales y los impuestos en el formulario de pedido de Bob.

Para ello agregue el siguiente código al final de la secuencia de comandos de PHP:

```
$totalqty = 0;
$totalqty = $tireqty + $oilqty + $sparkqty;

echo Items ordered: ‘.$totalqty.’<br/>';

$totalamount = 0.;

define('TIREPRICE'100);
define('OILPRICE!,10);
define('SPARKPRICE',4);

$totalamount = $tireqty * TIREPRICE
    + soilqty * OILPRICE
    + $sparkqty * SPARKPRICE;

echo ‘Subtotal:$’ .number_format($totalamount,3) .‘<br />';
$taxrate = 0.10; // el impuesto de ventas local es del 10%
$totalamount = $totalamount * (1 + $taxrate);
echo 'Total including tax: $ '.number_format($totalamount,2).'<br />';
```

Si actualiza la página n la ventana del navegador, verá un resultado similar al ilustrado en la figura 1.5

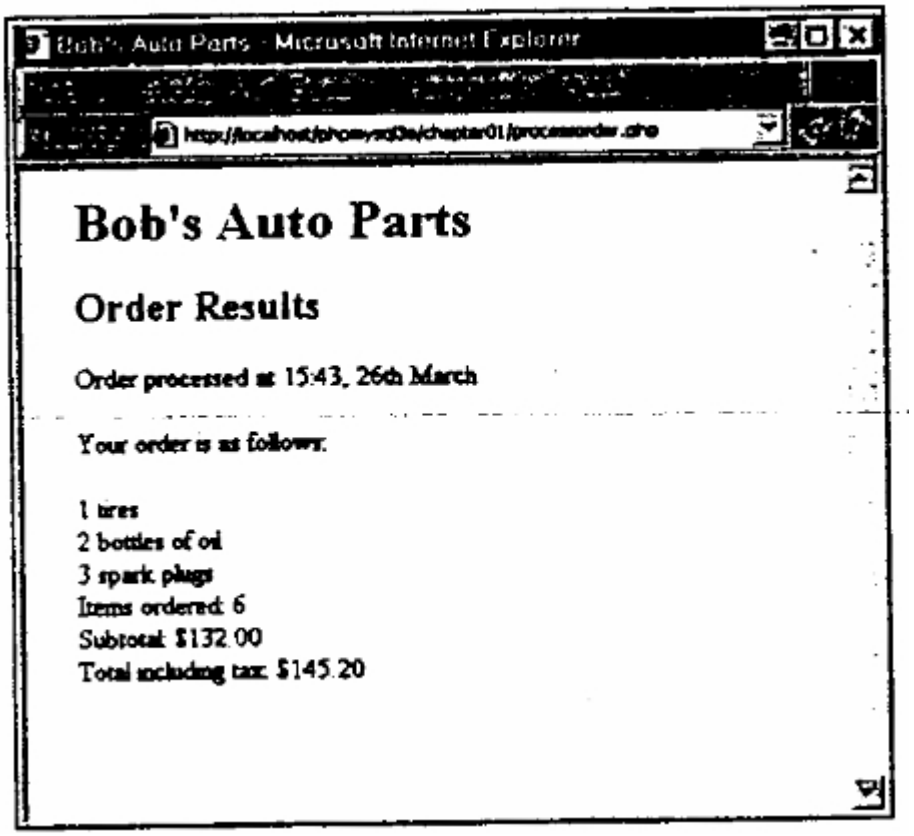

Figura 1.5 Se han calculado los totales del pedido de clientes, se les ha aplicado formato y se ha mostrado.

Como puede observar, en esta secuencia de código hemos utilizado varios operadores. El operador de suma (+) y el de multiplicación (*) se encargan de calcular las cantidades y el operador de concatenación (.) establece el resultado en el navegador. También hemos utilizado la función `number_format()` para aplicar formato a los totales como cadenas con dos decimales. Se trata de una función de la biblioteca de matemáticas de PHP.

Si se detiene a examinar los cálculos, es posible que se pregunte por qué se realizaron en ese orden. Por ejemplo, considere esta instrucción:

```
$totalamount = $tireqty * TIREPRICE
    + $oilqty * OILPRICE
    + $sparkqty * SPARKPRICE;
```

La cantidad total parece correcta pero, ¿por qué realizar las multiplicaciones antes que las sumas? La respuesta reside en la precedencia de los operadores, es decir, en el orden en el que se calculan.

Precedencia y asociatividad: evaluar expresiones

Por regla general, los operadores se evalúan siguiendo una precedencia u orden de prioridad fijado.

Los operadores llevan asignada una asociatividad, que es el orden en el que se evalúan los operadores con el mismo orden de prioridad. Suele ser de izquierda a derecha (o izquierda simplemente), de derecha a izquierda (o derecha simplemente) o no resulta relevante.

En la tabla 1.7 se recoge la precedencia de operadores y la asociatividad en PHP. En esta tabla, se incluyen los operadores de prioridad más baja en la parte superior y la precedencia va aumentando según se desciende por ella.

Tabla 1.7 Precedencia de operadores en PHP.

Asociatividad	Operadores
Izquierda	,
Izquierda	or
Izquierda	xor
Izquierda	and
Derecha	print
Izquierda	= += -= += /= .= %= &= |= ^= ~= <<= >>=
Izquierda	?:
Izquierda	||
Izquierda	&&
Izquierda	|
Izquierda	^
Izquierda	&
n/d	== != ===
n/d	< <= > >=
Izquierda	<< >>
Izquierda	+ - .
Izquierda	+ / %
Derecha	-! - ++ -- (entero) (doble) (cadena) (matriz) (objeto) @
Derecha	[]
n/d	New
n/d	()

Fíjese en que el operador con mayor prioridad no se ha analizado todavía: los paréntesis. El efecto de los paréntesis consiste en incrementar la prioridad de todos los elementos incluidos en su interior. Esto símbolos permiten alterar las reglas de prioridad cuando resulte necesario.

Recuerde la siguiente sección del último ejemplo:

```
$totalamount = $totalamount * (1 + $taxrate);
```

Si hubiéramos escrito

```
$totalamount = $totalamount * 1 + Staxrate;
```

el operador de multiplicación, que tiene precedencia sobre el operador de suma, se aplicaría en primer lugar, lo que nos daría un resultado incorrecto. El uso de los paréntesis permite forzar el cálculo de la subexpresión `1 + $taxrate` en primer lugar: Puede utilizar tantos paréntesis como desee en una expresión. En primer lugar se calcularán los más internos.

Utilizar funciones de variables

Antes de abandonar el mundo de las variables y de los operadores, vamos a examinar las funciones de variables de PHP. Se trata de una biblioteca de funciones que permite manipular y probar variables de distintas formas.

Probar y estableces tipos de varibales

La mayor parte de estas funciones se utilizan para probar el tipo de a variable. Las dos más generales son `gettype()` y `setttype()`. Éstas tienen los siguientes prototipos de funciones; es decir, lo que esperan los argumentos y lo que devuelve.

```
String gettype (mixed var);
Bool settype (mixed var, string type)
```

Para utilizar `gettype()`, la pasamos en una variable. Determinará su tipo y devolverá una cadena que contenga el tipo de nombre o “tipo desconocido” si no es uno de los tipos estándar, es decir, entero, doble, cadena, matriz u objeto.

Para utilizar `settype()`, le pasamos una variable cuyo tipo deseemos modificar y una cadena que contenga un nuevo tipo para dicha variable a partir de la lista anterior.

Nota

En este libro y en la documentación de php.net se hace referencia a un tipo de datos mixto. No existe como tal pero debido a la flexibilidad de PHP con respecto al control de tipos, muchas funcione se pueden adoptar muchos tipos de datos como argumento. Los argumentos para los que se permiten varios tipos de muestran con esta modalidad de pseudo tipo de datos.

Podemos utilizarla de la siguiente forma:

```
$a = 56;
echo gettype ($a).’<br />;
settype ($a, ‘double’);
echo gettype ($a).’<br />;
```

Cuando se llama a `gettype()` por primera vez, `$a` es de tipo entero. Tras llamar a `settype()`, el tipo se convierte en doble.

PHP también incorpora funciones para probar tipos. Cada una de estas toma una variable como argumento y devuelve `true` o `false`. Las funciones son:

- `is_array()`
- `is_double(), is_float(), is_real()` (Todas la misma función)
- `is_long(), is_int(), is_integer()` (Todas la misma función)
- `is_string()`
- `is_object()`
- `is_resource()`
- `is_null`
- `is_scalar()`: Comprueba si la variable es escalar, es decir, de tipo entero, booleano, cadena o flotante.
- `is_numeric()`: Comprueba si la variable es algún tipo de número o cadena numérica.
- `is_callable()`: Comprueba si la variable es el nombre de una función válida.

Probar el estado de las variables

PHP dispone de varias formas de probar el estado de una variable. La primera de estas formas es `iseet()`, que consta del siguiente prototipo:

```
bool iseet (mied var);
```

Esta función toma el nombre de una variable y devuelve `true` si existe y `false` en caso contrario. También puede pasar una lista de variables separadas por comas para que iseet() devuelva `true` si se han establecido todas las variables.

Puede eliminar una variable utilizando `unset()`. Éste es su prototipo:

```
void unset (mixed var);
```

Esta función suprime la variable pasada.

Por último, tenemos la función `empty()`. Esta función comprueba si existe una variable y si contiene un valor no vacío o distinto a cero, y devuelve `true` o `false` sagún caso. Su sintaxis es la sieguiente:

```
boolean empty (mixed var);
```

Vamos a examinar un ejemplo del uso de estas funciones.

Intente agregar el sieguiente código a su secuencia de comandos temporalmente;

```
echo 'isset($tireqty): ' . isset($tireqty).'<br />';
echo 'isset($nothere): ' . isset($nothere).'<br />';
echo 'empy($tireqty): ' . empty($tireqty).'<br />';
echo 'empy($nothere): ' . empty ($nothere).'<br />';
```

Actualice la página para ver los resultados.

La variable `$tireqty` debería devolver `1(tru)` de `isset()` con independencia del valor introducido o de si se introdujo algún valor en el campo del formulario. Que sea `emtpy()` o no depende del valor introducido.

La variable `$nothere` no existe, por lo que generará `false` desde `isset()` y trae desde `emtpy()`.

Estas funcione pueden resultar útiles para asegurarnos de que el usuario rellenó los campos apropiados del formulario.

Reinterpretar variables

Puede lograr el equivalente de convertir una variable llamando a una función. En este sentido, existen tres funciones útiles:

```
int intval(mixed var);
float doubleval (mixed var);
string strval (mixed val);
```

Cada una de éstas acepta una vairable como entrada y devuelve el valor de la variable convertida en el tipo apropiado. La función `intval()` también nos permite especificar la base de conversión para convertir una variable en una cadena, de esta forma podemos convertir cadenas hexadecimales en enteros.

Implementar estructuras de control

Las estructuras de control de un lenguaje permiten controlar el flujo de la ejecución de un programa o secuencia de comandos. Las estructuras de control se pueden agrupar en estructuras condicionales (o de bifurcación) y en estructuras de repetición, o bucles.

En las siguientes secciones se examinarán las implementaciones específicas de cada una de ellas en PHP.

Tomar decisiones con estructuras condicionales

Si deseamos responder lógicamente a las entradas de nuestros usuarios, nuestro código debe ser capaz de tomar decisiones. Esta función recae sobre las estructuras condicionales.

Instrucciones if

Podemos utilizar una instrucción `if` para tomar una decisión. Debemos darle una condición a la instrucción `if` para que la utilice. Si la condición fuera `true` se ejecutará el siguiente bloque de código. Las condiciones de las instrucciones `if` deben ir incluidas entre paréntesis.

Por ejemplo, si mandamos un pedido en el que no se incluyan neumáticos, latas de aceite ni bujías en el sitio de Bob, probablemente se deberá a que se ha pulsado accidentalmente el botón Submit Order. En lugar de indicarnos “Order processed”, la página podría devolver un mensaje mucha más útil.

Cuando el visitante realiza un pedido sin ningún artículo, podríamos indicárselo. Para ello podemos utilizar la siguiente instrucción `if`:

```
if ($totalqty == 0)
    echo ‘You did not order anything on the previous page!<br />’;
```

La condición que estamos utilizando es `$totalqty == 0`. Recuerde que el operador iguales `(==)` se comporta de manera distinta al operador de asignación.

La condición `$totalqty == 0` será trae por lo que `$totalqty` es igual a cero. Si `$totalqty` no es igual a cero, la condición será `false`. Cuando la condición sea `true`, la instrucción `echo` se ejecutará.

Bloques de código

A menudo necesitaremos ejecutar más de una instrucción dentro de una secuencia condicional como `if`. No es necesario colocar una nueva instrucción `if` para cada una de ellas. En su lugar, podemos agrupar un número de instrucciones en un bloque. Para declarar un bloque, enciérrelo entre llaves:

```
if ($totalqty == 0)
{
    echo ‘<font color=red>’;
    echo ‘You did not order anything on the previous page! <br />’;
    echo ‘</font>’;
}
```

Las tres líneas de código encerradas entre llaves forman ahora un bloque de código. Si la condición es `true`, se ejecutarán las tres líneas. Si la condición es `false`, se ignorarán las tres líneas.

Nota

Como se mencionó anteriormente, en PHP no es importante la disposición del código. Sin embargo, conviene sangrarlo por cuestiones de legibilidad. Las sangrías nos permiten distinguir de un vistazo qué líneas se ejecutarán sólo si se cumplen una condición, qué instrucciones se agrupan en bloques y qué instrucciones forman parte de bucles o funciones. En los ejemplos anteriores, se ha sangrado la instrucción que depende de la instrucción `if` y las instrucciones que forman el bloque.

Instrucciones else

Con frecuencia no sólo querrá decidir si desea que se ejecute una acción sino seleccionar una entre un conjunto de ellas.

Las instrucciones `else` permiten establecer la adopción de una acción alternativa cuando la condición de una instrucción `if` resulte `false`. Queremos avisar a los clientes de Bob si envían un pedido sin ningún artículo. Por otra parte, si realizan un pedido, en lugar de una advertencia, queremos mostrar les lo que han pedido.

Si reorganizamos nuestro código y agregamos una instrucción `else`, podemos mostrar un aviso o un resumen de los artículos solicitados.

```
if ($totalqty == 0)
{
   echo 'You did not order anything on the previous page!<br />';
}
else
{

   echo $tireqty . 'tires <br />'
   echo $oilqty . 'bottle of oil<br />';
   echo $sparkqty . 'spark plugs<br />';
}
```

Podemos desarrollar proceso lógicos más complejos anidando instrucciones `if`. En el siguiente código, no sólo se mostrará el resumen si la condición `$totalqty == 0` resulta ser cierta, sino que además cada línea del resumen sólo se mostrará si se cumple su propia condición.

```
if ($totalqty == 0)
{
   echo 'You did not order anything on the previous page!<br />';
}
else
{

   if ($tireqty>0)
         echo $tireqty . 'tires <br />'
   if ($oilqyt>0)
         echo $oilqty . 'bottle of oil<br />';
   if ($sparkqty)
   echo $sparkqty . 'spark plugs<br />';
}
```

Instrucciones elseif

Para muchas de las decisiones que tomamos suele haber más de dos opciones. Podemos crear una secuencia de varias opciones utilizando la instrucción `elseif`. Esta instrucción es una combinación de `else` e `if`. Al suministrar una secuencia de condiciones, el programa puede comprobar cada una de ellas hasta que encuentre una que se `true`.

Bob ofrece un descuento por grandes pedidos de neumáticos. La oferta se articula de la siguiente forma

- Menos de 10 neumáticos, sin decuento
- de 10 a 49, 5% de descuento
- de 50 a 99, 10% de decuento
- 100 o más de 100, 15 % de descuento

Podemos crear código para calcular el descuento utilizando condiciones e instrucciones `if` y `elseif`. Necesitamos utilizar el operador AND (&&) para comobinar las dos instrucciones en una.

```
if ($tireqty < 10)
      $discount = 0;
elseif ($tireqty >= 10 && $tireqty <=49)
      $discount = 5;
elseif ($tireqty >= 10 && $tireqty <=99)
      $discount = 10;
elseif ($tireqty >= 100)
      $discount = 15;
```

Fíjese en que da l o mismo escribir `elseif` o `else if`, ya que ambas son correctas.

Si va a escribir un conjunto de instrucciones `elseif` en cascada, debería ser consciente de que sólo se ejecutará uno dé los bloques o instrucciones. En este ejemplo no importa porque todas las condiciones son mutuamente excluyentes, sólo una puede ser verdadera a la vez. Si escribiéramos las condiciones de forma que más de una pudiera ser verdad era, sólo se ejecutaría el bloque o instrucción situada a continuación de la primera condición cierta.

Instrucciones switch

La instrucción `switch` funciona de una forma similar a la instrucción `if`, pero permite que la condición tome más de dos valores. En una instrucción `if`, la condición puede ser `true` o `false`. En una instrucción `switch`, la condición puede tomar cualquier número de valor es diferentes, siempre y cuando se evalúe en un tipo único (entero, cadena o doble). Es necesario incluir una instrucción `case` para cada valor al que desee reaccionar y, opcionalmente, una instrucción `case` predeterminada para procesar aquellos valores para los que no se hayan incluido una instrucción `case` especifica. Bob quiere saber qué tipo de publicidad atrae visitantes a su sitio. Para ello podemos agregar una pregunta a nuestro formulario de pedidos. Inserte el siguiente código HTML dentro del formulario de pedidos y el formulario presentará un aspecto parecido al ilustrado en la figura 1.6.

```
<tr>
   <td>How did you find Bob’s</td>
   <td><select name=”find”>
         <option value = “a”>I’m a regular customer
         <option value = “b”>TV advertising
```

```
        <option value = “c”> Phone directory
        <option value = “d”>Word of mouth
    </select>
    </td>
</tr>
```

Este código de HTML agrega una nueva variable de formulario cuyo valor será “a”, “b”, “c” o “d”. Podríamos procesar esta nueva variable con una serie de instrucciones `if` y `elseif` como la siguiente:

```
if ($find == ‘a’)
    echo ‘<p>Regular customer.</p>;
elseif ($find == ‘b’)
    echo ‘<p>Customer referred by TV advert.</p>;
elseif ($find == ‘c’)
    echo ‘<p>Customer referred by phone directory.</p>;
elseif ($find == ‘d’)
    echo ‘<p>Customer referred by word of mouth.</p>;
```

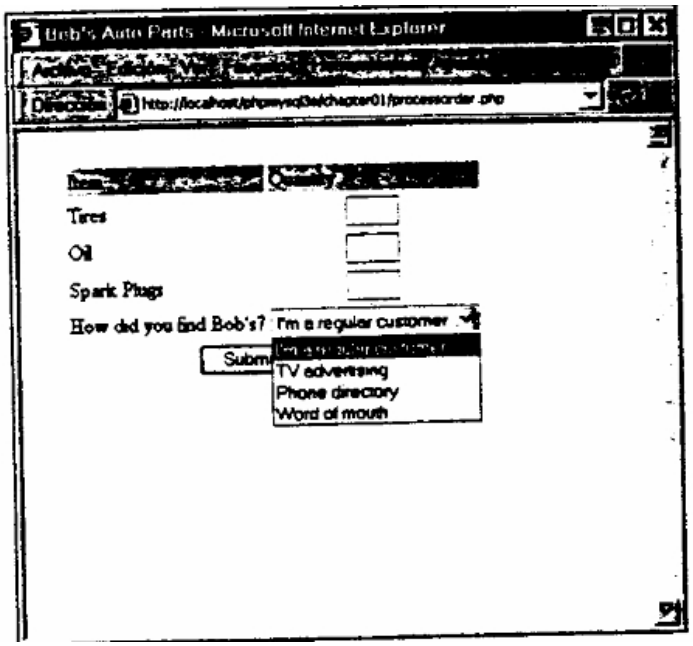

Figura 1.6 El Formulario de pedidos pregunta a los visitantes cómo encontraron el sitio Bob’s Auto Parts.

Como alternativa podríamos utilizar una instrucción `switch`:

```
switch($find)
{
    case ‘a’:
        echo’<p>Regular customer.</p>;
        break;
    case ‘b’:
        echo’<p>Customer referred by TV advert.</p>;
        break;
    case ‘c’:
        echo’<p>Customer referred by phone directory.</p>;
        break;
    case ‘d’:
        echo’<p>Customer referred by word of mouth.</p>;
```

```
        break;
    default :
    echo '<p>We do not know how this customer found us.</p>;
    break;
}
```

Apreciará que en ambos ejemplos se supone que ha extraído `$find` de la matriz `$_POST`.)

La instrucción `switch` se comporta de una forma ligeramente diferente a una instrucción if o `elseif`. Una instrucción `if` afecta sólo a una instrucción a menos que se utilicen llaves de manera deliberada para crear bloques de instrucciones. Una instrucción `switch` se comporta de forma contraria

Cuando se activa un caso en una instrucción switch, PHP ejecutará las instrucciones hasta que alcance una instrucción `break`. Sin una instrucción `break`, la instrucción `switch` ejecutaría todo el código situado detrás del caso que resultara ser cierto al alcanzar la instrucción `break`, se ejecutará la línea de código situada tras la instrucción `switch`.

Comparar condiciones diferentes

Si no está familiarizado con estas instrucciones, es probable que se esté preguntando cuál es la mejor.

Sin embargo, no podemos dar una respuesta exacta a esta pregunta. No hay nada que pueda hacer con una o varias instrucciones `else`, `elseif` o `switch` que no pueda hacerse con un conjunto de instrucciones `if`. Debería intentar utilizar aquella opción que le resulte más sencilla de leer. Con el tiempo irá aprendiendo a seleccionar una u otra.

Iteración: repetir acciones

Una de las cosas para las que los ordenadores han demostrado siempre ser buenos es para automatizar tareas repetitivas. Si hay algo que necesite hacer de la misma forma una serie de veces, puede utilizar un bucle para repetir partes de un programa. Bob quiere una tabla que muestre el coste de envío en función de la distancia a la que se está enviando el paquete. El coste se puede calcular con una sencilla fórmula. Queremos que la tabla de costes de envío se parezca a la ilustrada en la figura 1.7.

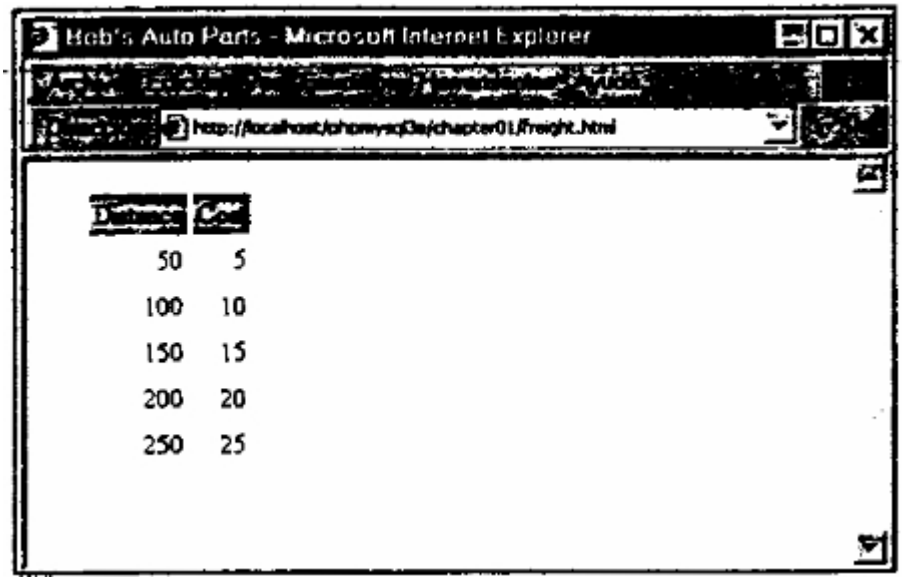

Figura 1.7. Esta tabla muestra el coste de envio en función de la distancia.

El listado 1.2 incluye el código HTML utilizado para mostrar esta tabla. Como observará, el código es largo y repetitivo.

Listo 1.2. freight.html. Tabla de costes de envío de Bob.

```
<html>
<body>
<table border="0" cellpadding="3">
<tr>
   <td bgcolor="#CCCCCC" align="center">Distance</td>
   <td bgcolor="#CCCCCC" align="center">Cost </td>
</tr>
<tr>
   <td align="right">50</td>
   <td align="right">5</td>
</tr>
<tr>
   <td align="right">100</td>
   <td align="right">10</td>
</tr>
<tr>
   <td align="right">150</td>
   <td align="right">15</td>
</tr>
<tr>
   <td align="right">200</td>
   <td align="right">20</td>
</tr>
<tr>
   <td align="right">250</td>
   <td align="right">25</td>
</tr>
</table>
</body>
</html>
```

Convendría delegar en un ordenador barato e incansable la tarea de escribir este código de HTML en lugar de recurrir a un humano que se aburriría con facilidad (y al que habría que pegar). Las instrucciones de bucle hacen que PHP ejecute una instrucción o un bloque de manera repetida.

Bucle while

El tipo de bucle más sencillo en PHP es el bucle `while`. Como en el caso de las instrucciones `if`, se basa en una condición. La diferencia entre un bucle `while` y una instrucción `if` es que éste ejecuta el siguiente bloque de código si la condición resulta ser `true`. Un bucle `while` ejecuta el bloque repetidamente mientras la condición sea `true`. Los bucles `while` se utilizan cuando no se sabe cuántas iteraciones resultarán necesarias para que la condición resulte cierta. si el número de iteraciones debe ser fijo, considere la posibilidad de utilizar el bucle `for`.

Las estructura básica de un bucle `while` es la siguiente:

```
while {condición} expresión;
```

El siguiente bucle mostrará los números del 1 al 5

```
$num = 1;
while ($num <= 5)
{
    echo $num.”<br />”
    $num++;
}
```

Al principio de cada operación, se prueba la condición. Si es falsa, el bloque no se ejecuta y el bucle finaliza. A continuación, se ejecutará la instrucción situada por detrás del bucle. Podemos utilizar un bucle `while` para realizar algo más útil como mostrar la tabla de costes de envío repetitivos de la figura 1.7. El listado 1.3 utiliza un bucle `while` para generar la tabla de gastos de envío.

listo 1.3. freight.php. Generación de la tabla de gastos de envío de Bob en PHP.

```
<body>
<table border=”0” cellpadding=”3”>
<tr>
    <td bgcolor=”#CCCCCC” align=”center”>Distance</td>
    <td bgcolor=”#CCCCCC” align=”center”>Cost </td>
</tr>
<?
$distance = 50;
while ($distance <= 250)
{
    echo “<tr>\n <td align”right”>$distance</td>\n”
    echo “ <td align=”right”>”. $distance / 10 . “</td>\n</tr>\n”;
    $distance += 50;
```

```
}
?>
</table>
</body>
```

Para que el código HTML generado por nuestra secuencia de comandos resulte legible, debe incluir nuevas líneas y espacios. Como se indicó anteriormente, los navegadores ignoran estos elementos pero resultan importantes para facilitar la lectura de los humanos. Con frecuencia necesitará examinar el código HTML si el resultado obtenido no es el esperado. En el listado 1.3 verá que algunas cadenas incluyen los caracteres `\n`. Si s e incluye dentro de una cadena encerrada entre comillas dobles, representa un carácter de nueva línea.

Bucles for y foreach

La forma en que utilizamos los bucles `while` anteriormente resulta muy común. En primer lugar establecimos un contador. Antes de cada interacción, probamos el contador en una condición. Al final de cada iteración, modificamos el contador.

Podemos escribir este estilo de bucle de forma más compacta utilizando un bucle `for`. La estructura básica de los contadores `for` es la siguiente:

```
for ( expresion1; condición; expresión2)
   expresión3;
```

- La `expresión1` se ejecuta una vez al principio. En este parámetro se suele establecer el valor inicial de un contador.
- La `condición` se prueba antes de cada iteración. si la expresión devuelve false, la iteración se detendrá. En este parámetro se suele probar el contador con respecto a un límite.
- La `expresión2` se ejecuta al final de cada iteración. En este parámetro se suele ajustar el valor del contador.
- La `expresión3` se ejecuta una vez por iteración. Esta expresión suele ser un bloque de código y contendrá el grueso del código de bucle.

Podemos rescribir el ejemplo del bucle `while` del listado 1.3 como un bucle `for`. El código PHP se convertirá en:

```
<?php
for ($distance = 50; $distance <= 250; $distance += 50)
{
   echo “<tr>\n <td align=’right’>$distance</td>\n”;
   echo “ <td align=’right’>”$distance / 10 . “</td>\n</tr>\n”;
}
?>
```

Tanto la versión `while` como la versión `for` funcionan de manera idéntica. El bucle `for` resulta un poco más compacto y ahorra dos líneas.

Ambos tipos de bucles son equivalentes. Ninguno de los dos es mejor o peor que el otro. En una situación dada, puede utilizar el que le resulte más intuitivo.

Como comentario al margen, hay que decir que se puede combinar una variable de tipo variable con un bucle `for` para procesar una iteración a través de una serie de campos de formulario repetitivos. Si, por ejemplo, tiene campos de formulario con nombres como `name1`, `name2`, `name3`, etc., puede procesarlos de la siguiente forma

```
for ($i=1; $i <= $numnames; $i++)
{
    temp = “name$i”;
    echo $$temp. ‘<br />’; // o cualquier procesamiento que desee realizar
}
```

Al crear dinámicamente los nombres de las variables, podemos acceder a cada campo uno por uno.

Además del bucle `for` existe el bucle `foreach`, diseñado específicamente para su uso con matrices. En un capítulo posterior comentaremos cómo utilizarlo.

Bucles do…while

El tipo final de bucle que mencionaremos se comporta de modo ligeramente diferente. La estructura general de una instrucción `do…while` es la siguiente:

```
do
    expresión;
while ( condición );
```

Un bucle `do..while` se diferencia de un bucle `while` en que la condición se prueba al final. Por lo tanto en un bucle `do…while`, la instrucción o el bloque incluido en el bucle se ejecuta siempre una vez al menos. Incluso si tomamos un ejemplo en el que la condición será `false` al principio y nunca puede ser `true`, el bucle se ejecutará una vez antes de comprobar la condición y el final.

```
$num = 100;
do
{
    echo $num. ‘<br />’;
}
while ($num < 1 );
```

Salir de una estructura de control o una secuencia de comandos

Si desea detener la ejecución de un fragmento de código, existen tres opciones que dependen del efecto que esté persiguiendo.

Si desea detener la ejecución de un bucle, puede utilizar la instrucción `break` como se comentó anteriormente en la sección sobre `switch`. Si utiliza esta instrucción en un bucle, la ejecución de la secuencia de comandos continuará en la línea situada tras el bucle. Si desea saltar hasta la siguiente iteración de bucle, puede utilizar la instrucción `continue`. Si desea terminar de ejecutar la secuencia de comandos PHP entera, puede utilizar la instrucción `exit`. Esta opción resulta útil al realizar tareas de comprobación de errores. Por ejemplo, podemos modificar nuestro ejemplo anterior de la siguiente forma:

```
if ( $totalqty == 0 )
{
   echo 'You did not order anything on the previous page!<br />';
   exit;
}
```

La llamada a `exit` impide que PHP ejecute el resto de la secuencia de comandos.

Utiliza una sintaxis alternativa de estructuras de control

Para todas las estructuras de control que hemos visto hasta el momento existe una sintaxis alternativa. Consiste en sustituir la llave de apertura (`{`) por dos puntos (`:`) y la llave de cierre con una nueva palabra clave como `endif`, `endswitch`, `endwhile`, `endfor` o `endforeach`, en función de la estructura de control utilizada. No existe una sintaxis alternativa para los bucles `do…while`.

Por ejemplo, al código

```
if ( $totalqty == 0 )
{
   echo 'You did not order anything on the previous page!<br />';
   exit;
}
```

se le podría aplicar esta sintaxis alternativa por medio de las palabras clave `if` y `endif`:

```
if ( $totalqty == 0 ):
{
   echo 'You did not order anything on the previous page!<br />';
   exit;
endif;
```

Utilizar declare

Otro estructura de control de PHP es la estructura `declare`, que no se utiliza con tanta frecuencia como otras estructuras de programación. El formato general que emplea es el siguiente:

```
declare (directiva)
{
    // bloque
}
```

Esta estructura se utiliza para establecer directivas de ejecución para bloques de código, es decir, para determinar la forma de ejecutar el código. En la actualidad, sólo se ha implementado una directiva de ejecución, `ticks`. Para establecerla debe añadir la directiva `ticks=n`. Le permite ejecutar una determinada función cada n líneas de código dentro del correspondiente bloque, lo que generalmente se utiliza para crear perfiles y para depuración.

Solo mencionamos la estructura de control `declare` a título informativo y en capítulos posteriores incluiremos algunos ejemplos de uso de las funciones `tick`.

Siguiente paso: guardar el pedido del cliente

Antes ya sabe cómo recibir y manipular pedidos de un cliente. En el siguiente capítulo examinaremos cómo almacenar el pedido para poder recuperarlo y servirlo.