Iies ntra. sra. de los remedios de ubrique

Tema 02 – Inserción de

2º DAW – Desarrollo Web Entorno Servidor

Juan Carlos Morer

CURSO 2024/2025

Tabla de Contenido

2 Inserción de Código en Páginas Web

El desarrollo de aplicaciones web del lado del servidor se fundamenta en la utilización de una serie de lenguajes y tecnología, que son ejecutadas por software especializado en uno o más servidores. En función del lenguaje que queramos utilizar, la distribución de la lógica de la aplicación o la forma de interactuar con la información que proviene del cliente, la gestión del flujo de trabajo de la aplicación y la configuración del entorno del servidor será diferente. En este segundo capítulo presentamos diferentes tipos de servidores web, introducimos las características principales de algunos lenguajes de servidor que permiten insertar código en páginas HTML y manejar el flujo de información para tratar la información enviada por un cliente a través de una página web

Objetivos:

- Reconocer la arquitectura de las aplicaciones web del lado del servidor dependiendo del lenguaje utilizado.
- Aprender a generar código de forma dinámica para ser mostrado por el cliente web.
- Conocer la sintaxis y les etiquetas propias de cada lenguaje de servidor que permite insertar código en páginas web ejecutadas en el servidor.
- Dominar la declaración de variables, tipos de datos simples, así como la conversión entre cada uno de ellos en función del lenguaje utilizado.
- Comprender la importancia de controlar el ámbito de declaración de cada variable y su influencia en el desarrollo de aplicaciones web.

2.1 Lenguajes y Tecnologías del Servidor

Tal y coma vimos en el primer capítulo, las tecnologías de servidor se basan en la existencia de un software especial denominado *servidor web*. La configuración de este componente software vendrá determinada por la utilización de un lenguaje u otro, ya que cada lenguaje requiere de una configuración determinada y de unos componentes específicos para ser ejecutado. Si querernos dominar el desarrollo de aplicaciones web dinámicas basadas en código de servidor es necesario comprender bien el escenario en el que se produce la ejecución de una aplicación web dentro de una arquitectura cliente/servidor tradicional

Lo primero es entender que el *servidor web* es un programa cuya misión última es servir datos en forma de documentos HTML (Hyper Text Markup Laguage) codificados en este lenguaje. El objetiva es proveer al cliente con textos complejos con enlaces, figuras, formularios, botones y objetos incrustados, tales como animaciones o reproductores de sonidos.

Para que los datos le lleguen al cliente es necesario tener en cuenta cómo se produce la comunicación entre un cliente y un servidor. El intercambio de datos entre ambos actores se hace por medio un protocolo determinado, generalmente mediante el protocolo HTTP. Por defecto, un servidor web se mantiene a la espera de peticiones HTTP realizadas por parte de un cliente (a través de un navegador web) "escuchando" un puerto de comunicaciones (normalmente el 80).

La secuencia de comunicación es la siguiente:

1. En un primer paso, el navegador solicita, como cliente DNS la traducción de una URL (por ejemplo *https://www.ieslosremedios.org*) a una IP.
2. Una vez que ha recibido la traducción del servidor DNS se realiza la petición HTTP al servidor web que tenga la IP obtenida (si ponemos directamente la dirección IP en vez de la URL en el navegador el resultado sería equivalente). Puesto que HTTP es un protocolo sin estado, cada petición de un cliente a un servidor no está influenciada por las transacciones anteriores. Esto quiere decir que el servidor tratará cada petición como una operación totalmente independiente del resto. Una novedad de este protocolo a partir de su versión 1.1 es que se pueden habilitar conexiones persistentes, lo cual afecta directamente a la eficiencia de las transacciones puesto que se permite enviar más objetos con un número menor de conexiones.
3. En el siguiente paso, el servidor procesa la solicitud realizada por el cliente y, tras ejecutar el código asociado al recurso solicitado en la URL, responde al cliente enviando el código HTML de la página.
4. Finalmente, el navegador del cliente, cuando recibe el código, lo interpreta y lo muestra en pantalla.

2.1.1 Protocolo

Conjunto de reglas y estándares que gobiernan el intercambio de información entre entidades dentro de una red. Existen muchos tipos de protocolos, cada uno con un propósito específico: FTP, POP3, SMTP, etc. Protocolo HTTP- Protocolo de la capa de aplicación usado en cada transacción web. Cada petición realizada mediante HTTP implica una conexión con el servidor que es liberada al término de la misma, es decir, no tiene estado. Por ejemplo, la solicitud de un archivo HTML con 5 imágenes requiere el establecimiento de 6 conexiones distintas (5 imágenes más la página HTML en sí).

2.1.2 Clasificación de Servidores Web

En función de cómo se procesen las peticiones realizadas por un cliente web podemos distinguir distintos tipos de servidores web. Las diferentes estrategias de optimización de la ejecución y de diseño del servidor dan lugar a diferentes tipos de servidores web:

2.1.2.1 Servidores basados en procesos.

Se puede considerar como la estrategia predecesora de todas las demás. Su funcionamiento se basa en la obtención de un paralelismo de ejecución mediante la duplicación del proceso de ejecución. Aunque existen diversas formas de gestionar los procesos, el más simple es aquel en el que el proceso principal espera la llegada de una nueva conexión (petición HTTP) y, en ese momento, se duplica creando una copia exacta (***fork***) que atenderá esta conexión, de tal forma que un proceso se dedica al tratamiento de la petición mientras que otro puede seguir escuchando nuevas peticiones. Esta estrategia admite múltiples optimizaciones como la técnica ***pre-fork***, incluida por el servidor Apache desde su versión 1.3.

2.1.2.1.1 Técnica Pre-fork

La idea es que en la memoria del servidor web lanzan varias copias del proceso que escucha peticiones. Estas copias de proceso se mantienen hasta que se necesitan. Entre las ventajas de esta técnica destaca la facilidad para ser implementada y la seguridad para aislar el procesamiento de una petición frente a otras. Por el contrario, el mayor inconveniente de esta

estrategia está en un mayor consumo de recursos de memoria (dedicada a cada proceso) y un menor rendimiento debido a que la gestión de los procesos es una tarea del sistema operativo.

2.1.2.2 Servidores basados en hilos

Frente a los servidores basados en procesos existe una alternativa más económica en cuanto consumo de recursos. En los servidores basados en hilos de ejecución, en lo que se refiere al procesamiento de las peticiones, el funcionamiento es básicamente idéntico al de los servidores basados en procesos. La mayor diferencia se encuentra en la distinción entre el concepto de hilo (***thread***) y el de proceso. La creación de un hilo por parte de un proceso no es tan costosa como la duplicación de un proceso completo. Los hilos de ejecución creados por un proceso comparten el mismo espacio de memoria reduciendo el consumo de memoria del servidor web de forma drástica. Este aspecto, si bien mejora el rendimiento del servidor, puede constituir un problema de seguridad a la hora de acceder al espacio de memoria. Puesto que, generalmente, todos los hilos creados por un servidor comparten la misma zona de memoria, si un hilo modifica una variable, el resto de hilos del mismo proceso verían esa variable con el valor modificado.

2.1.2.3 Servidores dirigidos por eventos

La novedad de este tipo de servidores os la utilización de ***sockets*** (espacios de memoria para la comunicación entra dos aplicaciones que permiten que un procese intercambie información con otro proceso estando los dos en distintas máquinas, en esta caso cliente y servidor). Las lecturas y escrituras sobre sockets son realizadas de forma asíncrona y bidireccional. De esta forma, aunque un cliente realizará peticiones de forma independiente (sin estado), el servidor podrá acceder a la información intercambiada con dicho cliente con solo acceder al sockets creado para tal cliente. La ventaja de este diseño radica principalmente en su velocidad. Por el contrario, el mayor inconveniente se encuentra en que la concurrencia de procesamiento es simulada; es decir, existe un solo proceso y un solo hilo, desde el cual se atienden todas las conexiones gestionadas por un conjunto de sockets.

2.1.2.4 Servidores implementados en el núcleo del sistema

La principal idea de esta estrategia de diseño es situar el procesamiento de cada una de las ejecuciones del servidor web en un espacio de trabajo perteneciente al sistema operativo (kernel) y no en un nivel de usuario (sobre el sistema operativo). Si bien es cierto que aunque esta alternativa permite acelerar el procesamiento de las peticiones al servidor, conlleva bastantes peligros. Cualquier incidencia (de seguridad o rendimiento) en este tipo de servidores implica un problema a nivel del sistema operativo que lo puede llegar a dejarlo completamente imperativo.

2.1.3 Servidores web más populares

La lista siguiente incluye la descripción de los servidores más utilizados en la actualidad. La gran mayoría de ellos soportan la ejecución de diferentes lenguajes de programación del lado del servidor, algunos de forma nativa (corno el soporte para ASP ofrecido por IIS) y otros mediante módulos y extensiones específicas (como el soporte para PHP de Apache). Estas extensiones hacen que el componente software que conforma cualquier servidor web sea capaz de responder adecuadamente a las peticiones que un cliente envía por HTTP a través de la Red. Por otro lado, en el capítulo anterior vimos cómo existían lenguajes que eran interpretados (corno PHP o ASP) y otros que eran precompilados y que necesitaban de un

substrato específico de ejecución para ser ejecutados (como ASP.Net o las aplicaciones basadas en Java). El soporte para uno u otro tipo de lenguajes vendrá determinado por los módulos y extensiones del servidor escogido. La configuración adecuada, por tanto, del núcleo del servidor web y de las correspondientes extensiones permitirá que el código desarrollado responda a los requisitos quo de él se esperan.

2.1.3.1 Apache Server

Se trata de un servidor HTTP diseñado para ser utilizado en múltiples plataformas y sistemas operativos. Entre sus características destacan el hecho de ser un servidor robusto, que implementa los últimos estándares y protocolos de la Red y que cuenta con gran capacidad de personalización y modularización . Además, al ser gratuito y de código abierto, cuenta con una comunidad de desarrolladores bastante amplia que hacen que exista gran documentación al respecto de su uso y configuración para diferentes propósitos. Puede configurarse tanto como servidor basado en hilos como basado en procesos.

Su diseño altamente flexible y configurable permite a los administradores de sitios web elegir qué características van a ser soportadas por el servidor, de tal forma que se pueden seleccionar qué módulos van a estar disponibles cuando se compile o ejecute el servidor web. Tiene módulos que ofrecen soporte para acceso a bases de datos, establecer páginas protegidas por contraseña, personalizar las páginas de error devueltas por el servidor, generar registros de actividad en múltiples formatos, etc.

2.1.3.2 Microsoft IIS

Es el servidor web de Microsoft según sus propias fuentes IIS (Internet Information Server) es un motor de páginas web flexible, seguro y fácil de gestionar que permite alojar cualquier tipo de contenido on- line. Destaca principalmente por dar soporte nativo a ASP (Active Server Pages) y a las diferentes tecnologías de la plataforma .Net. Además, permite añadir ciertos módulos para la ejecución de otros lenguajes como PHP. Este tipo de servidor lo llevan solo los sistemas basados en Windows, y suele instalarse como un complemento adicional al sistema operativo, accesible mediante permisos de administrador. En los sistemas Windows (versión Profesional), por ejemplo, IIS no viene instalado por defecto.

Al igual que el servidor Apache, permite configurar extensiones para reescritura de direcciones LTRL (para que varias CTRL sean respondidas por el mismo servidor), servicios de despliegue de aplicaciones multimedia, acceso a bases de datos o administración remota del servidor.

2.1.3.3 Sun Java SystemsWeb Server.

Se trata de un servidor web de alto rendimiento, de escalabilidad masiva y seguro que ofrece contenido dinámico y estático. Actualmente forma parte del software que Oracle tiene disponible para su descarga de forma gratuita (aunque el soporte técnico es de pago). Destaca por sus características de vitalización de dominios, versatilidad de configuración y seguridad robusta. Está optimizado para la integración y ejecución de aplicaciones Java (JSP, Servlets, NSAPI y DGI).

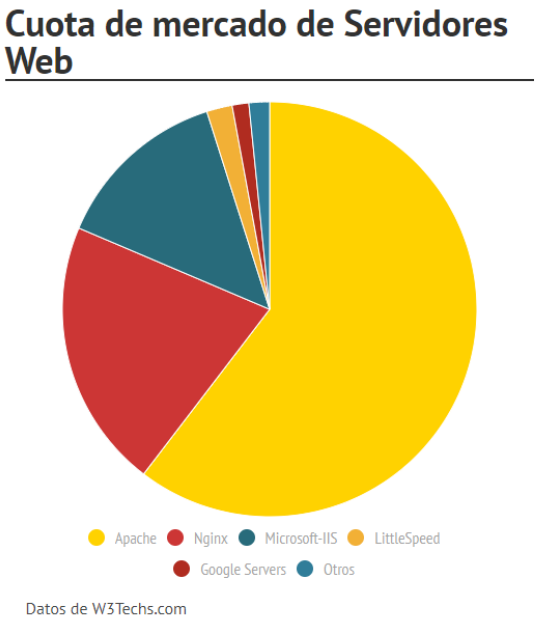
Como servidor web puede instalarse en prácticamente cualquier sistema operativo. Al igual que cualquier servidor de propósito general, admite la configuración de módulos y extensiones para la ejecución de código PHP, Ruby on Rails, Perl, Python y más lenguajes de servidor.

2.1.3.4 Nginx

Es un servidor HTTP que ha ganado cuota de mercado en los últimos años (de < 1% en 2007 a casi el 30% a finales de 2016). Es de código abierto y cuenta con una arquitectura modular de alto rendimiento, además de permitir funcionar corno servidor proxy para otros protocolos de Internet como IMAP o POP3. Nginx es un servidor asíncrono basado en hilos, conocido por su estabilidad, contener un amplio conjunto de filtros, ser fácilmente configurable y capaz de ser ejecutado en múltiples plataformas consumiendo relativamente pocos recursos. Tiene soporte para transacciones seguras basadas en SSL y TLS.

2.1.3.5 Lighttpd

Es un servidor web especializado para entornas en los que se requieren respuestas rápidas. También conocido como Lighttpd, soporta la mayoría de los estándares de comunicaciones de Internet y puede descargarse como software libre y de código abierto. Se distribuye bajo licencia BSD. Su consumo de memoria es bastante reducido (comparado con otros servidores web corno Apache) así como lo es su carga de procesamiento en la CPU. Está especialmente pensado para soportar un balanceo de cargas elevado sin perder prestaciones, utilizando poca RAM y poca CPU. Otra de las características de este servidor es que utiliza un solo proceso con varios hilos de ejecución pero sin tener la capacidad de crear nuevos hilos de ejecución en función de la demanda.



La siguiente sección se centra en Los principios básicos que un programador debe conocer para poder desarrollar una aplicación web ejecutada en cualquiera de los servidores mencionados con anterioridad. De esta forma, será posible adecuar la respuesta del servidor a las peticiones de un cliente web.

2.2 Obtención del código enviado al cliente

Los usuarios de páginas web normalmente se comunican utilizando un navegador web con un servidor con el fin de obtener una funcionalidad determinada. Esta funcionalidad puede abarcar desde la visualización de información (textual o multimedia) hasta la administración de un sitio web pasando por el almacenamiento de información, actualización de contenidos u otras operaciones más complejas. Con el fin de dar soporte a todas estas potenciales funcionalidades es absolutamente necesario ser capaces de ofrecer al cliente la información que necesita. La ejecución de código escrito en un lenguaje concreto por parte del servidor será responsable de generar dicha información.

En este capítulo nos centraremos en analizar las características fundamentales de PHP como ejemplo de lenguaje de programación web del lado del servidor PHP (Hypertext Preprocessor) es un lenguaje de scripting de propósito general y de código abierto que ha sido especialmente diseñado para el desarrollo de aplicaciones web y que puede ser embebido (intercalado) en código HTML. En capítulos sucesivos mostraremos los ejemplos de código correspondientes a PHP, ASP y JSP.

PHP nació en 1994 a partir de un conjunto de ficheros binarios que se utilizaban para recoger datos de tráfico de datos en sitios web. En 1997 el intérprete (parser) de este lenguaje fue reescrito con el fin de dar soporte a la generación dinámica de contenido web y de crear aplicaciones del lado del servidor. Desde su versión 4, (publicada en el año 2000), hasta su versión más actual, PHP se ha convertido en un lenguaje de gran popularidad entre la comunidad de desarrolladores web. Entre las razones de su éxito destaca una relativa facilidad para ser aprendido.

PHP sirve corno ejemplo paradigmático para comprender la arquitectura de un servidor web e identificar los componentes implicados durante el procesamiento de una aplicación web en respuesta a una solicitud del cliente. Las capas de una arquitectura genérica de un servidor que soporte la ejecución de PHP pueden verse en la Figura 2.1

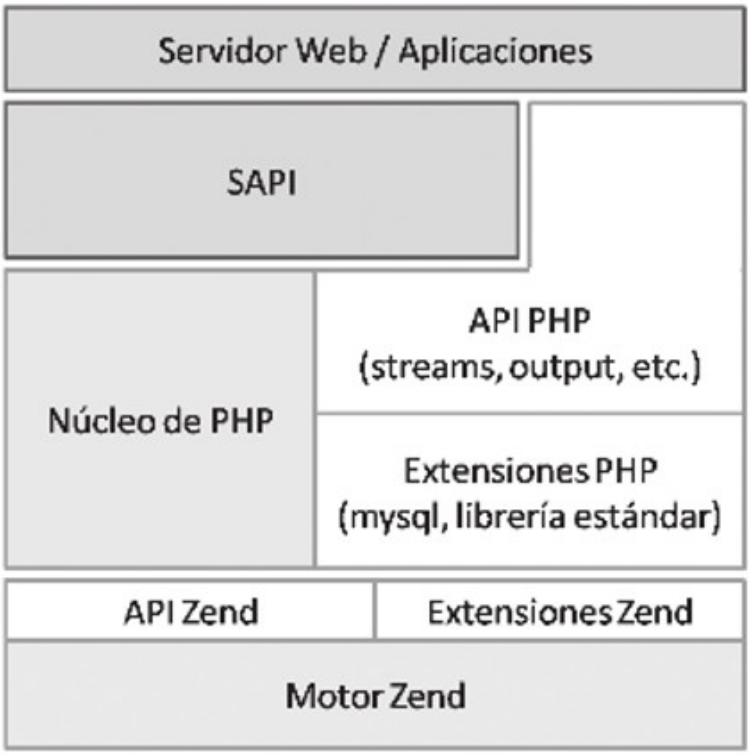


Figura 2.1. *Arquitectura genérica de PHP con Zend*

La capa más externa de esta arquitectura la conforma el propio servidor web, que es quien recibe las peticiones que el cliente hace a una dirección determinada. La capa inmediatamente inferior es la capa SAPI (Server Abstraction API) donde PHP interactúa con el servidor u otras aplicaciones. Esta capa SAPI gestiona, parcialmente, la inicialización y cierre de PHP como parte integrante de un servidor; contiene diferentes mecanismos para acceder a datos del cliente tales como las cookies o la información enviada por el método POST.

Por debajo de la capa SAPI se encuentra el motor de PHP propiamente dicho. El núcleo de PHP maneja la configuración del entorno de ejecución (asignando valores a variables globales y estableciendo las directivas de inicio del fichero de configuración php. ini) ofreciendo interfaces de programación (API PHP) tales como la interfaz de entrada/salida estándar, transformación de datos (data parsing) y, quizás lo más importante, ofreciendo un interfaz para cargar extensiones, tanto aquellas compiladas estáticamente como las cargadas dinámicamente.

En el corazón de la arquitectura de PHP se encuentra el motor Zend, que es el componente encargado del análisis y ejecución de las porciones de código PHP (scripts). El motor Zend, al igual que el de PHP, ha sido diseñado con el fin de admitir extensiones que permiten modificar completamente su funcionalidad (compilación, ejecución y manejo de errores) o parte de ella (cambiando las operaciones que se ejecutarán en caso de encontrarse un error).

Los pasos que se dan dentro del motor Zend cuando se ejecuta un script PHP son los siguientes:

1. El script es analizado por un analizador Léxico (lexer) que transforma el código escrito por el desarrollador en un conjunto de piezas (tokens) entendibles por la máquina. Estos tokens son pasadas después al analizador sintáctico (parser).
2. En el siguiente paso, el parser toma el conjunto de tokens y genera un conjunto de instrucciones (o código intermedio) que es ejecutado por el motor Zend. Este motor Zend actúa como máquina virtual que recibe como entrada el código intermedio de instrucciones y las ejecuta. Muchos parsers generan un árbol sintáctico que puede ser manipulado u optimizado antes de ser enviado al generador de código. El parser del motor Zend combina todos estos pasos en uno y genera el código intermedio directamente a partir de los tokens enviados por el analizador léxico. Este proceso se conoce con el nombre de compilación.
3. Después de que el código intermedio ha sido generado, la última fase es la ejecución donde interviene el componente ejecutor de Zend. Su función es ejecutar una por una las instrucciones indicadas en el código intermedio. Las fases de compilación y ejecución en el motor Zend se implementan internamente cono funciones que son accesibles desde el exterior por lo que es posible crear extensiones que modifiquen el comportamiento básico de cada una de estas fases.

El motor Zend funciona comoI una máquina virtual, es decir, se trata de un programa que simula una computadora fisica. El motor Zend implementa alrededor de 150 instrucciones específicas y optimizadas para la ejecución de código PHP (como inclusión de librerías o imprimir un string).

2.3 Etiquetas para la inserción de código

Como ya hemos comentado, la programación en el entorno del servidor, en el caso de los lenguajes embebidos, se centra en la inserción de código propio en páginas HTML. Cada uno de los lenguajes que se pueden utilizar para insertar código dentro de una página web utiliza una serie de etiquetas para delimitar los fragmentos de código que han de ser procesados por el servidor web. Independientemente del lenguaje utilizado (sea P. ASP, ISP u otros lenguajes de scripting de servidor) el componente del servidor encargado de procesar el código ignorará el código HTML que se encuentra fuera de dichas etiquetas.

El siguiente código muestra esta circunstancia:

```
<html>
	<head>
	<title>Primer ejemplo</title>
	</head>
	<body>
	<h1>Primer ejemplo: <br/></h1>
	<?php
		echo "Hola mundo.";
	?>
	</body>
</html>
```

El ejemplo anterior muestra el primer fragmento de código PHP de este capítulo, el archiconocido "Hola mundo'. Cuando el navegador web solicita este recurso, el fragmento de

código PHP se ejecuta en el servidor y el código HTML resultante es enviado al cliente. El resultado es el que podemos ver en la siguiente pantalla:

Obviamente, el script PHP mostrado en el ejemplo anterior es bastante sencillo y podríamos haber conseguido el mismo resultado editando el código HTML para incluir ese saludo directamente. Como podemos comprobar, el uso de PHP para insertar cadenas de caracteres estáticas en una página web es muy sencillo con PHP. Sin embargo, este sencillo ejemplo nos sirve para introducir algunas de las características básicas de los lenguajes de scripting de código embebido:

- Todo script comienza y termina con una etiqueta de inicio y otra de fin. En PHP, estas etiquetas son *<?php* y *?>* respectivamente en ASP se utilizan *<%* y *%>* y en JSP *<%=* y *%>*. En el caso de PHP también podemos usar *<?* y *?>* (denominadas short tags).
- Podemos configurar otros estilos de etiquetas en estos lenguajes. Por ejemplo, el estilo que utilizamos en HTML para JavaScript: *<script language="PHP">* y *</script >.*
- Los espacios en blanco que escribamos dentro del código embebido no tienen ningún efecto salvo para mejorar la legibilidad del código escrito. Cualquier combinación de espacios, tabuladores, retornos do carro y demás elementos usados para separar sentencias está permitida.
- El código de servidor embebido en páginas HTML está formado por un conjunto de sentencias que deben estar claramente separadas. En PHP y JSP esta separación se realiza con un punto y coma (;) mientras que en ASP esta separación se hace mediante retornos de carro.
- Los scripts embebidos pueden situarse en cualquier parte del recurso web ejecutado y puede ser intercalado en cualquier fragmento de HTML. El número de scripts que podemos tener dentro de un fichero HTML es indefinido.
- Cuando se ejecuta un código embebido, el script entero se sustituye por el resultado de dicha ejecución, incluidas las etiquetas de inicio y fin.

Para ilustrar estas características, podemos observar el siguiente fragmento de código que intercala código PHP en diferentes partes de un fichero HTML. En este caso podemos ver cómo hay tres fragmentos de código. El primero define el contenido de una variable ($salida) y los otros dos lo que hacen es escribir el valor de dicha variable entre el código HTML y en la posición en la que se encuentran los scripts PHP. Podemos ver cómo el valor asignado a la variable la utilizamos para indicar tanto el título de la página como el contenido de la misma.

```
<html>
<?php
	$salida = "Contenido PHP";
?>
```

```
<head>
	<title><?php echo $salida; ?></title>
<head>
<body>
	<h1>Segundo ejemplo:</h1>
	<?php
		echo $salida;
	?>
</body>
</html>
```

Si bien es cierto que una de las múltiples ventajas del código embebido es precisamente la posibilidad de situar los scripts en cualquier parte del fichero HTML, esta circunstancia también puede resultar en un mantenimiento más difícil de la aplicaciónweb desarrollada. Con el fin de disminuir los efectos de la dispersión del código de PHP podemos utilizar directivas de inclusión de código que veremos en capítulos posteriores.

2.3.1 Inclusión de Comentarios

Otra forma de mejorar la legibilidad del código y que se recomienda siempre como una buena práctica a la hora de programar, es la inclusión de comentarios explicativos intercaladas en el código.

En PHP, podemos incluir los comentarios utilizando un estilo fácilmente reconocible en otros lenguajes de alto nivel. Entre ellos destacamos los siguientes:

Comentario de una línea

```
// esto es un comentario de una línea
# esto es otra forma de comentario de una línea
```

Comentario en varias líneas

```
/* esta es la forma de
escribir comentarios de
varias líneas */
```

Comentar el código que escribimos se considera una buena práctica en el mundo de la programación. De esta formar podemos hacer descripciones básicas del código escrito que puede llevar a confusión. Además, hacemos que el código desarrollado sea más fácil de mantener y compartir.

2.3.2 Mostrar contenido en Páginas HTML

Como hemos visto en los primeros ejemplos, una de las sentencias más habituales a la hora de mostrar el contenido en una página HTML es la instrucción *'echo'* que sirve tanto para cadenas de caracteres como para imprimir variables. Sin embargo, no es la única alternativa ya que también podemos utilizar la sentencia *'print'* como vemos en los siguientes ejemplos:

```
echo "ejemplo de impresión de cadena de caracteres";
// el resultado de usar print sería el mismo
```

```
print "ejemplo de impresión de cadena de caracteres”;
// también se puedan imprimir números directamente
echo 215062;
// y mostrar el contenido de una variable
echo $variableResultado;
print $variableResultado;
```

La diferencia entre *echo* y *print* es que la sentencia echo puede imprimir más de un argumento, de la siguiente manera:

```
echo “imprimir primer argumento",“ y el segundo";
print “valor 1”, “valor 2”; // No correcto uso dos argumentos
print “valor 1”.“valor 2”; //Correcto uso el operador de concatenar
```

Otra forma de mostrar el contenido de una variable es utilizando el símbolo igual junto a la etiqueta de inicio sin necesidad de usar *echo* o *print*:

```
<?=$variableResultado; ?>
```

Las sentencias *echo* y *print* son funciones que pertenecen a la librería estándar de PHP y, por ello, también podemos usarlas con los parámetros entre paréntesis:

```
echo “mensaje”;
// es lo mismo que
echo ( “mensaje”);
```

Cuando utilizamos paréntesis con *echo* y *print* es necesario tener en cuenta el número máximo de parámetros admitidos por cada una de estas sentencias. En el caso de *echo*, el máximo es 1.

De cualquier forma, ambas sentencias pueden mostrar como resultado la combinación de cadenas de caracteres estáticas, números, vectores (arrays) y otros tipos de variables que comentaremos más adelante en este capítulo.

Con respecto a las cadenas de caracteres debemos hacer una mención especial al manejo que PHP hace de las *comillas simples* y *dobles* cuando queremos que se muestren (o no) como resultado en el HTML enviado al cliente.

Tanto las comillas simples (‘ ‘) como dobles (“ ”) se pueden usar para crear cadenas de caracteres:

```
echo 'Este texto se mostrará';
echo "exactamente igual que este”;
```

Si queremos mostrar comillas simples (o dobles) como parte de la salida, lo que debemos hacer es utilizar comillas dobles (o simples) como delimitadores de la cadena de caracteres de la siguiente forma:

```
// Ejemplo de cadenas de caracteres con comillas
echo "This string has a ‘: a single quote!";
echo ‘This string has a ": a double quote’;
```

Otra forma de evitar que las comillas se consideren delimitadores "*escaparlas*", es decir, indicarle al intérprete de PHP que el carácter a continuación de la barra invertida *(\)* es un carácter que debe imprimirse sin interpretarlo.

```
// Ejemplo de caracteres "escapados"
echo "Este texto contiene una    \” (una comilla doble) “;
echo 'Este texto contiene una \’ (una comilla simple)';
```

Si querernos incluir otros caracteres especiales dentro de una cadena de caracteres delimitada por dobles comillas que vamos a insertar en el HTML que se le enviará al cliente., utilizaremos también la barra invertida. De esta forma: si queremos insertar una barra invertida en el texto de salida escribiremos \ \ y lo mismo sucedería con el símbolo*$ (\$).*

```
echo "Este texto contiene una barra invertida: \\";
echo " y este un símbolo del dólar: \$ “;
```

Los ejemplos anteriores sirven para ilustrar la forma básica de insertar contenido estático o dinámico obtenido por la ejecución de cierto código en el lado del servidor utilizando PHP como lenguaje de scripting. Otros lenguajes de servidor como ASP o JSP tienen sentencias y funciones similares que permiten obtener el mismo resultado.

2.4 Variables

Los datos manejados por el código de cualquier lenguaje de programación web se almacenan en variables. Estas variables son almacenes temporales de datos que permiten gestionar los datos utilizados por la aplicación web durante el flujo de ejecución de una página determinada. Como hemos podido comprobar en la sección anterior, toda variable tendrá asociado un tipo de datos.

2.4.1 Definición y Uso

Las variables en PHP se identifican por el símbolo del dólar *($)* seguido del nombre de una variable. En PHP, las variables no necesitan ser declaradas explícitamente y, además, no tienen un tipo definido hasta que no se les asigna un valor. Si observamos el siguiente fragmento de código, le asignamos el valor 15 (de tipo integer) a la variable *$var*. A partir de ese momento, *$var* será de tipo integer:

```
$var = 15;
```

Puesto que la variable del ejemplo anterior la usamos para asignarle un valor, es en ese mismo instante cuando se declara implícitamente la variable (se reserva espacio de memoria para almacenar dicho valor). Esta circunstancia es común a todas las variables PHP: la primera vez

que se usan, su tipo es definido (o redefinido) y la variable es declarada. Como vimos en la sección anterior, el tipo de esta variable puede cambiar a lo largo de su vida:

```
$var = 15;
$var = "cambio de tipo";
```

Lo que sucede durante la ejecución de las sentencias anteriores es que el tipo de la variable *$var* cambia de integer a string conforme cambia el contenido que almacenan. Esta característica de PHP es una de sus mayores fortalezas y es lo que lo convierte en un lenguaje denominado *Loosely typed* (debilmente tipado), es decir, que no requiere de una definición explícita de las variables ni de su tipo. Esta circunstancia hace que el uso de PHP sea más flexible pera también conlleva ciertos riesgos.

El nombre de una variable en PHP tiene que cumplir una serie de reglas:

- El nombre debe comenzar con una letra o con un guión bajo ("_").
- El nombre únicamente puede contener caracteres alfanuméricos y guiones bajos (a-z, A-Z, 0-9 y _).
- El nombre de una variable no debe contener espacios en blanco. Si queremos formar el nombre de una variable con más de una palabra lo que se suele hacer es utilizar un guión bajo entre ellas *($mi_variable)* o poner la primera letra de cada palabra en mayúsculas *($miVariable)*.

Existen técnicas de 'ofuscación' de código cuyo objetivo es realizar un cambio no destructivo que complique la lectura o comprensión del mismo por motivos diversos (por ejemplo dificultar la ingeniería inversa). Una forma básica de ofuscación es utilizar nombres de variables sin sentido creados a partir del nombre de una palabra reservada levemente.

Un aspecto importante del uso de variables en PHP es que es *case-sesitive* (sensitiva a las mayúsculas). Esto quiere decir que las variables *$variable, $Variable y $VARIABLE* son variables completamente diferentes.

Cometer un error con el nombre de una variable puede tener consecuencias graves tales como bucles que nunca terminan. En el contexto de una página web que tuviera este tipo de problemas, sería que el navegador web devolvería un error de *timeout* porque el servidor no sería capaz de terminar la ejecución de la página solicitada. Podemos ver un ejemplo en el siguiente fragmento de código:

```
for ($contador=0; $contador<10; $Contador++)
miFuncion( );
```

La variable *$contador* nunca se incrementa, al contrario lo que sucede es que otra variable, *$Contador*, es incrementada cada vez que se ejecuta el bucle. De este modo *$contador* siempre es menor que 10 con lo que nunca se sale del bucle. Entre los errores más habituales que podemos encontrarnos están cambios en una letra del nombre de una variable, que letras estén en mayúsculas o minúsculas, omitir guiones bajos o simples errores tipográficos.

2.4.2 Tipos de Datos y Variables

Como ya vimos en secciones anteriores, podemos incluir una variable como parte de una cadena de caracteres. PHP transformara el contenido existente entre unas dobles comillas por texto y reemplazará el nombre de las variables con el valor que tengan asignado en el momento de ejecutar dicha sentencia. Podemos verlo en el siguiente ejemplo:

```
$hora = 12;
$minutos = 15;
$ciudad = "Sevilla";
$texto = "El tren destino $ciudad sale a las $hora:$minutos";
echo $texto;
// la salida será El tren destino Sevilla sale a las 12:15
```

En PHP podemos diferenciar tipos:

- Escalares (*boolean*, *integer*, *float* y *string*)
- Compuestos (*array* y *object*)
- Especiales (*Null* y *Resource*)
- Pseudotipos (*mixed*, *number* y *callback*)..

Las variables que son de tipo escalar pueden contener un único valor cada vez. Las variables de tipo compuesto, como arrays y objects, están formadas por múltiples valores escalares u otros valores compuestos. Cuando una función puede admitir (o devolver) parámetros de diferente tipo, podemos utilizar el pseudo-tipo mixed para indicarlo.

A continuación describiremos más en detalle las características de estos tipos de datos:

2.4.2.1 Tipo Boolean

Las variables de. tipo]malean son las más sencillas: únicamente pueden contener los valores *true* (cierto) y *false* (falso), tal y como vemos en los siguientes ejemplos:

```
$repetidor = false;
$test = true;
```

2.4.2.2 Tipo Integer y float

Una variable que almacena un tipo integer, contiene un valor entero (positivo o negativo) mientras que una variable de tipo float contendrá números reales con una parte entera y una parte decimal (separado por un punto).

```
//Esto, es un integer
$num = 7;
//Esto es un float
$var2 = 5.12;
```

Una variable que contenga un tipo float también puede representarse utilizando una notación exponencial:

```
// Esto es un float que representa al número 12340
```

```
$var3 = 1.234e4;
// Esta variable contiene un valer equivalente a 0.04
$var4 = 4e-2;
```

2.4.2.3 Tipo String

Las cadenas de caracteres (tipo string) ya las vimos cuando introdujimos las funciones echo y print.

Veamos dos ejemplos más:

```
$variable = "Esto es una cadena da caracteres";
$test = 'Esto también es un string':
```

2.4.2.4 Definición valores constantes

PHP también permite la definición de valores constantes cuyo contenido no varía durante la ejecución del código. La forma de definir constantes en PHP es mediante la función *define()* que admite dos argumentos obligatorios: un nombre y un valor que se asociará a partir de ese momento a ese nombre. El siguiente ejemplo ilustra esta circunstancia:

```
Define("PI", 3.14159);
echo PI;
// La salida seria 3.14159
```

Aunque para el uso del número PI, PHP posee ya la constante M_PI y la función PI() tal y como se muestra en el siguiente ejemplo

```
<?php
echo pi(); // 3.1415926535898
echo M_PI; // 3.1415926535898
?>
```

Podemos observar cómo las constantes no llevan el símbolo $delante del nombre. Además, su valor no puede cambiarse una vez han sido definidas, pueden ser accedidas desde cualquier fragmento del fichero PHP en ejecución y la única restricción acerca de su valor es que únicamente pueden contener valores escalares (ni arrays ni objects).

2.4.3 Asignación de valores a las variables

Hasta ahora, hemos visto ejemplos muy sencillos de asignación de valores a variables utilizando el signo "=". La mayoría de las asignaciones numéricas y expresiones que se dan en otros lenguajes de alto nivel también funcionan en PHP. Veamos algunos ejemplos:

```
// Sumar enteros para producir un entero
$var = 2 + 2;
// Resta, multiplicación y división
// que dan como resultado un entero o un valor
// dependiendo del valor inicial de la variable
$var = (($var - 3) * 4) / 2;
```

```
// Varias formas de añadir 1 a la variable $var
$var = $var + 1;
$var += 1;
$var++;

//Lo mismo poro con la resta
$var = $var - 1;
$var -= 1;
$var--;

// Duplicar un valor
$var = $var * 2;
$var *= 2;

// Dividirlo por 2
$var = $var / 2;
$var /= 2;
```

Con respecto a la asignación y manejo de cadenas de caracteres, PHP se comporta de manera igualmente sencilla:

```
// Asignar un valor de texto a una variable
$var = "test string";

// Concatenar 2 cadenas
// para producir un resultado unificado
$var = "cadena" . "unida";

// Alladir un texto al final de una cadena de caracteres
$var = "cadena";
$var = $var . "unida";

// El operador de concatenación
// tambin tiene su versión corta
$var .= "test";
```

2.4.4 Variables no inicializadas

Si intenta utilizar una variable antes de asignarle un valor, se genera un error de tipo E_NOTICE, pero no se interrumpe la ejecución del script. Si una variable no inicializada aparece dentro de una expresión se calcula tomando el valor por defecto para ese tipo de dato.

```
<?php
	$var1 = 100;
	$var3 = 200 + $var2;
	echo $var3; // muestra 100
	$var3 = 100 * $var2;
	echo $var3; //muestra 0

¿>
```

2.4.5 Conversiones entre tipos de datos

En la sección anterior hemos visto cómo se codifica el contenido que finalmente visualizaremos en el navegador del cliente. La ejecución de una sentencia *echo* o *print* siempre

dará, como resultado, una cadena de caracteres. Esto se debe a que el contenido de un documento HTML siempre se codifica en modo texto. Sin embargo, cuando se trabaja con cualquier lenguaje de programación de alto nivel es habitual que los lenguajes permitan la identificación y utilización de diferentes tipos de datos, además de las consabidas cadenas de caracteres.

La conversión entre los diferentes tipos de datos es, además, algo habitual cuando se trabaja en entornos web. Por ejemplo, a la hora de enviar la información al cliente, el resultado finalmente mandado no será el mismo si lo que se manda es una fecha que si es un número real. Además, las operaciones más habituales con datos requieren que los operadores sean del mismo tipo, de ahí la importancia de conocer los procesos de conversión de datos.

PHP ofrece diferentes mecanismos que permite transformar las variables de un tipa en otro. Algunas de ellas son:

- *string strval (mixed variable*)
- *integer intval (mixed variable)*
- *float floatval(mixed variable)*

Además existen funciones específicas de gran utilidad que permiten establecer el tipo de una variable. Es el caso de la función:

- *settype (mixed variable, string type)* donde el parámetro 'variable' admite valores de tipa array, boolean, float, integer, object o string y el parámetro ‘type’ es una cadena de caracteres que indica el tipo en el que queremos transformar el parámetro variable.

PHP soporta también el cambio de tipo de datos de forma similar a cómo se hace en lenguaje C. Para ello, podemos escribir entre paréntesis el tipo deseado justo delante del nombre de una variable, de tal forma que cuando se ejecute esa sentencia, el tipo resultante de la variable será el indicado entre paréntesis. La siguiente tabla muestra un resumen de este tipo de conversiones:

Tabla 2.1 Conversiones implícitas de tipo de datos en PHP

Sentencia	Resultado
`(int) $var` `(integer) $var`	Conversión a tipo `integer`.
`(bool) $var` `(boolean) $var`	Conversión a tipo `boolean`.

`(float) $var`
`(double) $var`
`(real) $var`
	Conversión a tipo `float`.

`(string) $var`	Conversión a tipo `string`.
`(array) $var`	Conversión a tipo `array`.
`(object) $var`	Conversión a tipo `object`.

Las reglas de conversión de datos (*cast rules*) son en su mayoría de sentido común, sin embargo, hay algunas conversiones que no son tan intuitivas. La siguiente tabla muestra cómo diferentes valores asignadas a una variable denominada $var son convertidos utilizando los operadores de conversión *(int), (bool), (string) y (float).*

Tabla 2.2 Ejemplos de conversión de datos en PHP

Valor de $var	(int) $var	(bool) $var	(string) $var	(float) $var
null	0	false	""	0
true	1	true	"1"	1
false	0	false	""	0
0	0	false	"0"	0
3.8	3	true	"3.8"	3.8
"0"	0	false	"0"	0
"10"	10	true	"10"	10
"6 metros"	6	true	"6 metros"	6
"hola"	0	true	"hola"	0

Otra peculiaridad de la conversión de tipos de datos en PHP se da cuando combinamos en una misma expresión dos variables que inicialmente tienen tipos diferentes o cuando pasamos una variable como argumento a una función que espera un tipo de dato diferente. En estos casos en los que una variable de un tipo se utiliza como si fuera de otro tipo, PHP convierte automáticamente la variable a un valor del tipo requerido (obteniéndose un resultado como el mostrado en la Tabla 2.2).

Veamos algunos ejemplos que muestran qué ocurre cuando sumamos enteros (*integer*) o reales (*float*) a cadenas de caracteres o cuando estos son concatenados a cadenas de caracteres:

```
// $var se convierte a tipo integer con valor 35
$var = "20" + 15;

// $var se convierte a tipo float con valor = 35.1
$var = "20" + 15.1;

// $var se convierte a string con valor = "20 años"
$var = 20 . " años";
No todas las conversiones de datos son tan obvias y en ocasiones son el motivo
de numerosos errores en el código difíciles de detectar:

// $var sa convierte a tipo integer con valor = 20
// Observese la diferencia da usar "." Ó "+"
$var = 20 + " años";

// $var se convierte a tipo integer con valor = 42
$var = 40 + "2 razones";

// $var se convierte a tipo float...
// ...pero no tiene ningún sentido el primer operando
```

```
$var = "prueba" * 4 + 3.14159
```

La conversión automática de tipos debido a la utilización de ciertos operadores puede cambiar el tipo de una variable. Es lo que ocurre en los siguientes ejemplos:

```
$var = "1"; //$var es un string con valor = "1"
$var += 2;    //$var Ahora es un integer con valor = 3
$var /= 2;    //$var ahora es un float con valor = 1.5
$var *= 2;    //$var sigue siendo un float con valor = 3
```

Debemos tener un cuidado especial cuando manejemos valores no boolean como si fueran de ese tipo. Esta situación se da en muchas ocasiones porque algunas de las funciones de la librería estándar devuelven un valor "false" (de tipo boolean) cuando el valor resultado no puede devolverse. Es importante que tengamos en cuenta que un valor devuelto como 0, 0.0, "0", un string vacío, null o un array vacío se interpreta siempre como "false" cuando se utilizan como valores boolean.

La solución a este tipo de problemas pasa por preguntar, siempre, por el tipo de la variable devuelta como resultado de la invocación de una función. Para ello, PHP ofrece una serie de funciones que permiten comprobar cuál es el tipo de una variable:

- *boolean is_int (mixed variable)*
- *boolean is_float (mixed variable)*
- *boolean is_bool (mixed variable)*
- *boolean is_string (mixed variable)*
- *boolean is_array (mixed variable)*
- *boolean is_object (mixed variable)*

Como podemos ver, todas las funciones anteriores devuelven un valor booleano "cierto" (true) o "falso" (false) para la variable pasada como argumento dependiendo de si dicha variable coincide o no con el nombre de la función que se está invocando. Por ejemplo, el siguiente fragmento de código imprime el valor 1 ya que la variable $prueba pasada como argumento contiene un valor de tipo float.

```
$prueba = 13.0;
echo is_float ($prueba);
// imprime 1 (significa true)
```

Cuando estamos desarrollando aplicaciones web con PHP, en muchas ocasiones nos interesa ver por pantalla cuál es el tipo de una variable determinada. Para ello, PHP nos ofrece las funciones print_r () y var_dump(), que imprimen el tipo de una variable y su valor asignado en ese momento.

```
$variable = 15;
var_dump ($variable);
// La salida será: int 15.
```

A modo de resumen, no debemos olvidar que la información manejada por los lenguajes de programación web del lado del servidor puede ser de naturaleza muy diversa. Al igual que en cualquier lenguaje de programación, la distinción de diferentes tipos de datos es un aspecto crucial a la hora de desarrollar aplicaciones.

2.4.6 Precedencia de los operadores

La precedencia de operador en una expresión es similar a la precedencia definida en cualquier otro lenguaje: la multiplicación y la división se realizan antes que la resta y la suma, etc. Sin embargo, depender de la precedencia de operador en ocasione nos lleva a escribir un código farragoso y difícil de entender que a menudo resulta en un aumento del número de errores. Por ello, lo que se recomienda normalmente es utilizar paréntesis para romper la precedencia de operador y clarificar qué operaciones se realizarán antes que otras. Esto se debe a que los paréntesis tienen la máxima precedencia:

Tomemos por ejemplo la siguiente expresión:.

```
$operacion = 2 + 5 * 6;
```

A la variable $operacion le asignarnos el valor 32 porque la multiplicación tiene más prioridad que la suma. Sin embargo, el resultado se ve mucho más claro si se utilizan paréntesis:

```
$operacion = 2 + (5 * 6);
```

2.4.7 Estado de una variable

En PHP tenemos diferentes funciones para comprobar el estado de una variable independientemente del tipo de datos que almacene. Las tres funciones más populares y frecuentemente utilizadas para este fin son ***isset()***, ***is_null()*** y ***empty()***. Las tres devuelve un valor lógico true/false y en muchos casos utilizar cualquiera de ellas puede ser indiferente, en otros podemos encontrarnos con situaciones inesperadas.

En general, se suele entender ***is_null()*** como opuesta a ***isset()***, y esta como opuesta a ***empty()***. Pero esto no es del todo cierto y en este apartado se explican las peculiaridades de cada una de estas funciones para poder utilizarlas de forma correcta.

- *Boolean is_null(mixed var)*
- *boolean isset(mixed var)*
- *boolean empty(mixed var)*

Is_null()

Compruba si una variable es NULL

Por tanto, esta función devolverá *true* cuándo la variable comprobada tenga un valor NULL (nulo). En PHP NULL se considera un valor especial que representa a una variable sin valor, lo que puede ocurrir cuándo la variable no haya sido definida, cuándo esté definida pero sin valor asignado, cuándo se le asigne el valor NULL o cuándo haya sido destruida con usent().

```
//La variable no está definida
// Devuelve true y lanza un aviso tipo Notice: undefined variable
var_dump( is_null($var) );

//Se asigna la constante NULL
$var = NULL;
var_dump( is_null($var) );

//Se declara la variable pero no se le asigna ningún valor
$var;
var_dump( is_null($var) );

//La variable es destruida con unset()
$var = "Hola";
unset($var);
var_dump( is_null($var) );
```

En los cuatro casos anteriores *is_null($var)* devuelve true. Además, en el primer caso cuándo *$var* no ha sido definida, y en el último caso cuándo *$var* ha sido destruida, se lanzará un error tipo Notice: Undefined variable. Por el contrario, si se utilizan variables no definidas con *isset()* y *empty()* no aparecerá ningún tipo de aviso.

Isset()

La función *isset()*, según el manual PHP, **determina si una variable ha sido declarada y su valor no es NULO**. Es importante fijarse bien: si una variable tiene valor nulo, aunque haya sido declarada, *isset()* devolverá false. En todos los ejemplos anteriores para *is_null()*, *isset()* devolvería false mientras que *is_null()* devolvió true, por eso se consideran funciones opuestas. También es importante fijarse que *isset()* devuelve true para variables con valores vacíos, por ejemplo, cadenas de texto vacías:

```
//isset() devuelve false por que $var no ha sido definida
var_dump( isset($var) );

//isset() devuelve false aunque la variable haya sido declarada pues //su
valor es NULL
$var;
var_dump( isset($var) );

//isset() devuelve true. El valor ya no es nulo aunque esté vacío
$var = "";
var_dump( isset($var) );
```

Empty()

Determina si una variable está vacía.

La definición es muy sencilla pero abarca muchos posibles escenarios. Una variable se considera vacía si no existe (aquí coincide con *isset()* y con *is_null()*) o si su valor es *false*. El valor *false* para una variable puede ser un valor lógico false, una cadena vacía, el número 0

(cero), un array vacío, el valor NULL y, aquí la peculiaridad, el string "0" (no como valor numérico, sino como cadena de texto) también dará positivo en la función *empty()*.

```
//La variable no está definida. empty() devuelve true
var_dump( empty($var) );

//empty() devuelve true. El valor de $var es nulo
$var;
var_dump( empty($var) );

//empty() devuelve true para un string vacío
$var = "";
var_dump( empty($var) );

//empty() devuelve true para un array vacío
$var = array();
var_dump( empty($var) );

//empty() devuelve true para el string "0"
$var = "0";
var_dump( empty($var) );

//empty() devuelve true para el número entero 0
$var = 0;
var_dump( empty($var) );

//empty() devuelve true para el valor lógico false
$var = false;
var_dump( empty($var) );

//empty() devuelve false para el valor lógico true
$var = true;
var_dump( empty($var) );
```

Una variable puede ser "destruida" utilizando la función *unset()*

- *unset (mixed var [, mixed var [, ...]])*

Después de invocar la función *unset()* en el siguiente ejemplo, la variable ''$var" estará en un estado “no definida”:

```
$var = “valor”;
unset($var);

// La siguiente sentencia no imprime nada
if (isset($var)) echo “var está definida”;
```

Otra forma de comprobar si una variable está vacía es forzar a que su tipo sea boolean (utilizando el operador (bool) indicada con anterioridad) y comprobar si se interpreta a true o a false. El ejemplo interpreta la variable $var como de tipo boolean, que es equivalente a comprobar si $var no está vacía con !empty ($var)

```
$var = “valor”;
// Ambas sentencias imprimirán lo mismo
```

```
if ((bool)$var)     echo "var no está vac1a";
if (¡empty($var)) echo "var no está vacia";
```

La Tabla 2.3 muestra los valores de retorno para la ejecución de las funciones isset() y empty() y la aplicación del operador de conversión (bool) . Podemos observar como PHP devuelve algunos resultados extraños como que empty() devuelve true cuando le asignamos el valor "0".

Tabla 2.3 Comprobación del estado de una variable

Contenido de $var	isset($var)	empty($var)	(bool) $var
$var = null;	false	true	false
$var = 0;	true	true	false
$var = true	true	false	true
$var = false	true	true	false
$var = "0";	true	true	false
$var = "";	true	true	false
$var = "foo";	true	false	true
$var = array( );	true	true	false
unset ($var);	false	true	false

¿Cuándo utilizar cada una?

Esta es una pregunta con muy difícil respuesta y depende de cada situación:

- Se usa empty() con mucha más frecuencia que las demás y se se usa cuando necesito comprobar que una variable no tiene un valor vacío pero no es estrictamente necesario comprobar que la variable haya sido declarada. Si la variable no existe empty() devuelve true pero no lanza error. Si la cadena de texto "0" es un valor válido y posible para la variable a comprobar no se debe utilizar empty() por considerar este valor como vacío.
- Se usa isset() cuando se necesita comprobar que una variable existe y no es nula aunque tenga un valor vacío.
- Se usa is_null() para pocos casos y son aquellos en los que es necesario comprobar específicamente que una variable sea nula o no exista.
- Se se necesita hacer una comparación estricta pero no se si la variable ha sido declarada, se comprueba previamente con isset().

2.4.8 Ámbito de una variable

Un aspecto importante a la hora de manejar variables en las aplicaciones web de servidor es su ámbito. Se entiende por ámbito al contexto dentro del que la variable está definida, es decir, la zona del programa en la que pueda ser accedida o utilizada. De esta forma, es posible hablar

de ámbito local o ámbito global. Además, existe el concepto de variable de sesión que veremos en un capítulo más avanzado.

Las variables utilizadas dentro de una función son diferentes de aquellas que se utilizan fuera de ella. De tal forma que las variables internas a una función única y exclusivamente pueden ser utilizadas dentro de dicha función. El siguiente ejemplo ilustra esta circunstancia:

```
function duplicar($var) {
       $temp = $var * 2;
}
$variable = 5;
duplicar($variable);
echo "El valor de la variable \$temp es: $temp";
```

Este ejemplo devolvería la siguiente salida

```
El valor de la variable $temp es:
```

.. y ningún valor para $temp. En este caso se dice que el ámbito de la variable $temp es local a la función duplicar () y se descarta (se destruye) al salir de dicha función.

Como podemos observar, el motor de PHP no devuelve ningún mensaje de error asociado a no encontrar la variable $temp. Simplemente asume que la variable está vacía. Sin embargo, una de las características configurables de las extensiones para PHP de los servidores web es que podemos indicar al motor de PHP que muestre un aviso acarea de aquellas variables no declaradas.

Si lo que queremos es utilizar un valor que es local para una función en cualquier otra parte del código, la forma más fácil que tenemos de hacerlo es devolviendo el valor de la variable. Nuestro ejemplo anterior quedaría de la siguiente forma:

```
function duplicar($var){
       $resultado = $var * 2;
       return ($resultado);
}
$variable = 5;
$temp = duplicar($variable);
echo "El valor de la variable \$temp es: $temp";
```

Este ejemplo devolvería la siguiente salida:

```
El valor de la variable $temp es = 10
```

Podríamos haber utilizado el nombre de la variable $temp dentro de la función duplicar(). Sin embargo, la variable $temp interna a la función sería diferente de la que utilizaríamos para recoger el valor devuelto por la función. La regla general nos dice que las variables utilizadas

exclusivamente dentro de las funciones son locales a esas funciones, independientemente de si existe o se usa una variable que tiene un nombre idéntico fuera de esas funciones.

Corno veremos en capítulos posteriores, existe una excepción a esta regla: las variables definidas “por referencia” y aquellas variables declaradas como “globales” en la función no son locales a dicha función.

Si queremos utilizar la misma variable en cualquier parte del código, incluido dentro de las funciones, podemos servirnos de la sentencia “*global*”. Esta sentencia declara que una variable dentro de una función es la misma que la variable que hemos utilizado (o utilizaremos) fuera de esa función. Veamos un ejemplo:

```
function duplicar( ) {
       global $temp;
       $temp = $temp * 2;
}
$temp = 5;
duplicar ();
echo "El valor da la variable \$temp es: $temp”;
```

Puesto que $temp ha sido declarada dentro de la función como variable global, dicha variable se entenderá como una variable global que puede ser accedida desde fuera de la función duplicar(). Como consecuencia de declarar la variable $temp como global, la salida del script anterior será la siguiente:

```
El valor de la variable $temp es: 10
```

Debemos indicar, no obstante, que la utilización de variables globales sin control puede resultar un código difícil de mantener y propenso a dar errores. Por otro lado, existen diferencias en cómo debemos declarar una variable global en PHP y en otros lenguajes de scripting de servidor. En PHP la calificación de global de una variable se realiza dentro de la función donde la utilizaremos. En ASP y JSP es al contrario, la calidad de global se indica en el programa principal o fuera del ámbito de la función.