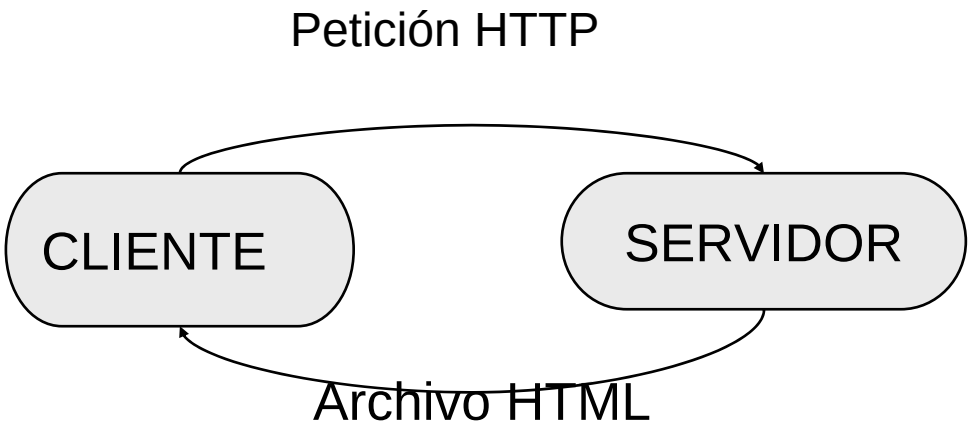
Programación en lenguaje PHP
Nivel Avanzado

1. Introducción e instalación

1. Introducción e instalación
Funcionamiento de un servidor Web

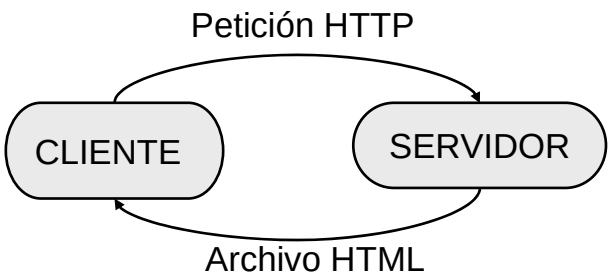
En su versión más sencilla, un servidor Web responde a una petición de usuario enviando páginas HTML que tiene almacenadas por medio del protocolo HTTP.

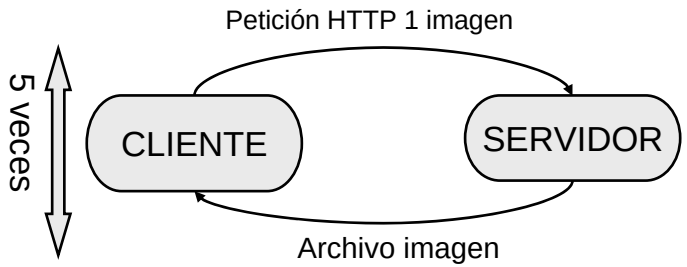


1. Introducción e instalación
Funcionamiento de un servidor Web

Si la página tiene varios elementos de imagen, se deberá establecer una conexión HTTP para enviar cada imagen.

Por ejemplo, una página con cinco imágenes precisará un total de 6 conexiones HTTP.





1. Introducción e instalación
Funcionamiento de un servidor Web

Si la página es dinámica puede ocurrir que el dinamismo esté en la parte cliente.

En este caso el funcionamiento no cambia mucho.

Como un elemento más de la página se envía, por ejemplo, el applet Java, archivos Javascript.

Luego el elemento dinámico se ejecuta en la máquina cliente, dotándola de dinamismo visual, pero no de contenido.

1. Introducción e instalación
Funcionamiento de un servidor Web

Si, en cambio, optamos por dinamismo en la parte del servidor, la cosa cambia más.

Aunque desde el punto de vista del usuario no cambia mucho, pues él solicita una página y recibe HTML como respuesta.

La página que se envía al cliente, sin embargo, ya no se envía tal cual está almacenada en el servidor, sino que se construye en el momento de ser enviada.

1. Introducción e instalación
Funcionamiento de un servidor Web

La construcción de esta página se hace ejecutando un script que la página tiene escrito en su interior.

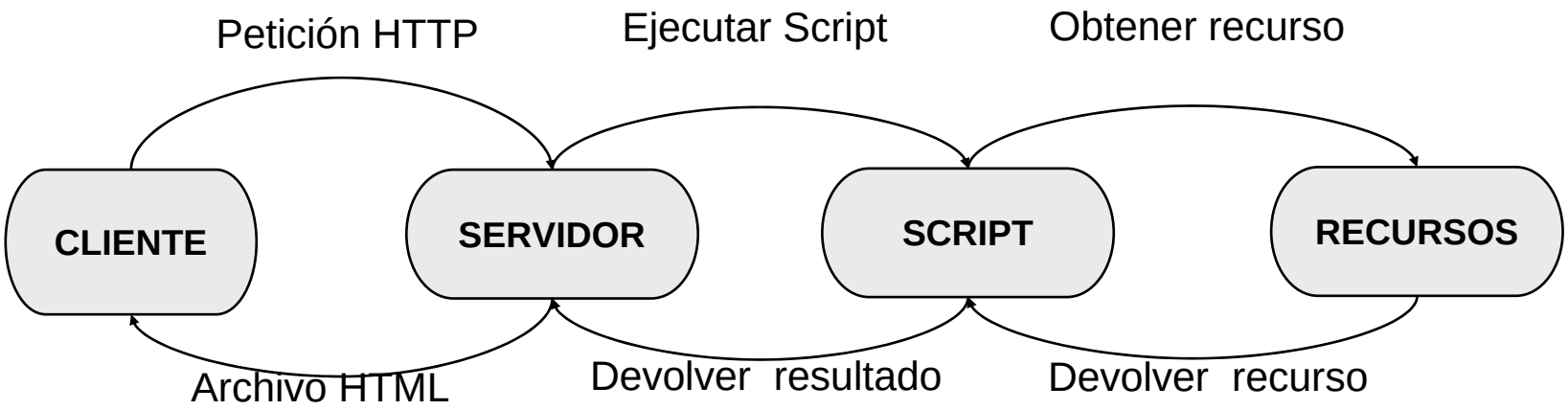
El servidor Web no ejecuta el script por si mismo, sino que se lo envía a un módulo que es capaz de interpretarlo.

Este módulo a su vez puede usar otros recursos, como ficheros de datos, bases de datos, etc. en la configuración de la página.

Ya compuesta sólo de HTML, la página es entregada de nuevo al servidor, quien la envía al cliente.

1. Introducción e instalación
Funcionamiento de un servidor Web

El proceso en su conjunto quedaría representado por la siguiente figura.



1. Introducción e instalación
Historia de PHP

PHP/FI Fue creado por Rasmus Lerdorf en 1995 como un simple conjunto de scripts para acceder a su trabajo on line.

A medida que fue necesitando más funcionalidad acabó por escribir una implementación en C, que finalmente liberó para que cualquiera pudiera utilizarla.

1. Introducción e instalación
Historia de PHP

PHP/FI 2.0 Apareción en 1997 como una segunda escritura de PHP.

Esta versión contó ya con numerosas colaboraciones de otros programadores.

Esta versión salió ya del ámbito personal y hasta 50.000 sitios de Internet llegaron a tenerlo instalado, a pesar de su corta vida.

1. Introducción e instalación
Historia de PHP

PHP 3.0 era la primera versión que se parecía fielmente al PHP tal y como lo conocemos hoy en día. Fue creado por Andi Gutmans y Zeev Zuraski en 1997 rescribiéndolo completamente

Una de las mejores características de PHP 3.0 era su gran extensibilidad. Además de proveer a los usuarios finales de una sólida infraestructura para muchísimas bases de datos, protocolos y APIs.

Muchos desarrolladores se unieron y enviaron nuevos módulos de extensión.

1. Introducción e instalación
Historia de PHP

PHP 4 partió una reescritura del núcleo que buscaba mejorar la ejecución de funciones complejas y la modularidad del código base.

El nuevo motor (llamado Zend) se introdujo a mediados de 1999 y fue liberado en Mayo de 2000.

Además de las mejoras en la ejecución, PHP 4 proporciona soporte para numerosos servidores Web, sesiones HTTP, buffers de salida, y numerosas nuevas instrucciones.

1. Introducción a PHP
Historia de PHP

El 13 de julio de 2004, fue lanzado **PHP 5**

Utilizando el motor Zend Engine II

Incluye las ventajas que provee el nuevo Zend Engine 2: Mejor soporte para la Programación Orientada a Objetos, mejoras de rendimiento, mejor soporte a XML, mejor soporte para MySQL, …

1. Introducción e instalación
PHP frente a otros lenguajes

Velocidad: No solo la velocidad de ejecución, la cual es importante, sino además no crear demoras en la máquina. PHP se integra muy bien junto a otro software, especialmente bajo ambientes Unix, cuando se configura como módulo de Apache, está listo para ser utilizado.

Estabilidad: PHP utiliza su propio sistema de administración de recursos y dispone de un sofisticado método de manejo de variables, conformando un sistema robusto y estable.

1. Introducción e instalación
PHP frente a otros lenguajes

Seguridad: PHP provee diferentes niveles de seguridad, estos pueden ser configurados desde el archivo .ini

Simplicidad: Los usuarios con experiencia en C y C++ podrán utilizar PHP rápidamente.

1. Introducción e instalación
La instalación de PHP

En términos generales la instalación de PHP va a constar de tres pasos:

1. La instalación de un servidor Web (si no hay ya uno instalado), que en nuestro caso será Apache.
2. El desempaquetado de la distribución PHP que vayamos a usar.
3. La configuración del servidor Web para que pueda usar el intérprete PHP.

Veremos a continuación estos tres pasos en detalle.

1. Introducción e instalación
Instalación de Apache

Puedes obtener el programa en su sitio web, `http://www.apache.org/`.

La versión binaria de Apache para Windows viene en formato MSI, es un instalador de Microsoft que ya viene de serie a partir de Windows 2000.

Para comenzar la instalación de los archivos basta hacer doble click en el archivo *.msi* que hemos bajado.

1. Introducción e instalación
Instalación de Apache

Como vamos a usar Apache como servidor de pruebas pondremos *localhost* en el nombre del servidor.

El nombre del dominio y el correo electrónico no tienen importancia.

Elegiremos a continuación la instalación completa.

1. Introducción e instalación
Instalación de Apache

Lo siguiente es elegir el directorio de instalación. Para no tener que manejar una trayectoria tan larga, lo cambiamos por `c:\wamp\apache22`

El instalador creará automáticamente una carpeta llamada Apache22.

Una vez acabada la instalación el servidor web estará instalado y funcionando como un servicio de windows denominado Apache2.2

1. Introducción e instalación
Instalación de PHP

Paso 1:

Podemos obtener PHP desde la página www.php.net. Descomprimimos el fichero bajado (p.ej. `php-5.2.6-Win32.zip`), en un nuevo directorio

Podemos colocar PHP como un subdirectorio de la instalación WAMP. Así que en el WinZip elegimos `c:\wamp` como destino.

WinZip creará un directorio con un nombre largo que luego renombraremos `php526`.

1. Introducción e instalación
Instalación de PHP

Paso 2:

Renombramos el archivo `php.ini-recommended` como `php.ini`.

Copiamos `c:\wamp\php526\php5ts.dll` al directorio `c:\windows`.

1. Introducción e instalación
Preparar Apache para PHP

Existen dos formas de instalar PHP para que funcione con Apache:

Como CGI: En este caso Apache ejecuta PHP como un programa independiente cada vez que necesita interpretar una página PHP.

Es menos eficiente, pero se considera más estable y seguro.

Como módulo de Apache: Las rutinas de PHP pasan a ser un módulo más de Apache, al que este accede mediante llamadas a la correspondiente dll.

1. Introducción e instalación
Preparar Apache para PHP

Instalación de PHP como CGI. Paso 1.

Tenemos que editar el fichero `httpd.conf` de Apache, que se encuentra en el subdirectorio `conf` de la instalación.

Buscamos la sección `alias_module`.

Introducimos la siguiente línea nueva:

```
ScriptAlias /php/ "C:/WAMP/php526/"
```

1. Introducción e instalación
Preparar Apache para PHP

Instalación de PHP como cgi. Paso 2.

A continuación buscamos la sección `mime_module` y escribimos:

```
AddType application/x-httpd-php .php
```

A continuación escribimos:

```
Action application/x-httpd-php /php/php-cgi.exe
```

Dotamos a Apache de permisos en el directorio de PHP:

```
<Directory “C:/WAMP/PHP526”>
    Allow from all
</Directory>
```

1. Introducción e instalación
Preparar Apache para PHP

Instalación de PHP como cgi. Paso 3.

Para comprobar que funciona hacemos lo siguiente:

Abrimos nuestro editor de texto y escribimos:

```
<?php  phpinfo(); ?>
```

guardamos este archivo como `prueba.php` dentro del **document root** de la instalación de Apache que es donde se guardan los documentos a servir.

A continuación arrancamos Apache y el navegador, donde pondremos la dirección

```
http://localhost/prueba.php
```

1. Introducción e instalación
Preparar Apache para PHP

Instalación de PHP como módulo Apache. Paso 1.

En lugar de las líneas que añadíamos a nuestro archivo `.conf` de apache, añadimos las siguientes:

```
LoadModule php5_module c:/wamp/php526/php5apache2_2.dll
AddType application/x-httpd-php .php
PHPIniDir “c:/wamp/php526”
```

1. Introducción e instalación
Instalación MySQL

Paso 1.

Podemos obtener el servidor MySQL en la dirección http://dev.mysql.com/downloads/ . En el momento de confeccionar este manual la última versión es la 5.0.67 .

Con un poco de paciencia, se obtiene el archivo mysql-5.0.67-win32.zip de unos 45.3 MB.

1. Introducción e instalación
Instalación MySQL

Paso 2.

Tras descomprimir el .zip en una carpeta temporal, se ejecuta `setup.exe`.

Las pantallas del programa de instalación no ofrecen ninguna dificultad: bienvenida, tipo de instalación (típica, completa o personalizada), directorio de instalación y resumen de las opciones seleccionadas; antes de la copia de archivos.

Finalmente una ventana nos informa de que la instalación ha finalizado y nos pregunta si deseamos configurar el servidor.

1. Introducción e instalación
Instalación MySQL

Paso 3.

En la configuración seleccionamos el tipo de configuración (detallada o estándar), el uso de la máquina (desarrollo, servidor, …), ajustes del tipo de BD (multifunción, transaccional, OLAP, ayuda a decisión), puerto de comunicaciones, juego de caracteres, instalar como servicio, contraseña de administrador, …

1. Introducción e instalación
Instalación MySQL

Paso 4.

Ya hemos instalado MySQL y está funcionando mediante un servicio.

Para que las extensiones mysql y mysqli funcionen es necesario que la librerial ***libmysql.dll*** se encuentre en el PATH (una posible solución es copiarla en la carpeta de Windows).

1. Introducción e instalación
Instalación MySQL

Paso 5.

Vamos a utilizar un cliente de MySQL realizado con PHP denominado **PHPMyAdmin** descargado de http://www.phpmyadmin.net/home_page/downloads.php .

Hay que descomprimir el fichero *phpMyAdmin-2.11.8.1-all-languages-utf-8-only.zip* y situar su contenido dentro del sitio web.

Hay que configurar el servidor, lo que se puede hacer de forma gráfica en la opción ***Setup Script*** del cliente *http://localhost/phpMyAdmin*

2. Programación orientada a objetos

2. P.O.O.
Declaración de clases

En PHP se declaran clases con la sentencia *class*

```
class nombre {
// ...definición de variables
// ....
// ...constructores (opcional)
// ....
// ...definición de funciones
// ....
}
```

2. P.O.O.
Declaración de clases

Las declaraciones de variables dentro de una clase deben ir precedidas de la ***visibilidad*** .

Las variables pueden inicializarse en el momento de ser declaradas escribiendo el signo “=“ y el valor.

```
class ejemplo {
  public $variable1=0;
  private $variable2; }
```

2. P.O.O.
Declaración de clases

El constructor es una función que se llama __construct() y se ejecuta cada vez que se crea un nuevo objeto de la clase.

Las funciones de la clase o *métodos* se declaran de la forma habitual con la palabra *function.*

Para referirse a las variables de la propia clase se debe usar la palabra *$this* de la siguiente forma:

2. P.O.O.
Creación de objetos

Una vez declarada una clase se crea un nuevo objeto con la instrucción *new.*

Esta instrucción crea el nuevo objeto y devuelve un identificador, que podremos guardar en una variable.

```
class reloj {
// ....
}

$mi_reloj = new reloj
```

2. P.O.O.
Creación de objetos

Una vez creado un objeto podemos acceder a sus variables y métodos usando su identificador y el operador ->.

Por ejemplo, si la clase reloj tiene un método llamado *poner_en_hora* y una variable llamada *hora*, se accedería a ellos de la siguiente forma:

```
$mi reloj -> poner_en_hora();
echo $mi_reloj -> hora;
```

2. P.O.O.
Herencia de objetos

Se puede crear una nueva clase a partir de otra ya existente utilizando la palabra clave *extends*.

```
class reloj_pulsera extends reloj {.......}
```

Esta sentencia crea una clase con todos los métodos y variables de la clase a la que extiende, *reloj*, a la que se añadirían nuevas características propias de esta clase.

2. P.O.O.
Herencia de objetos

El constructor de la clase padre -si existe- no se ejecuta automáticamente al crear un nuevo objeto del tipo de la clase extendida, por lo que habría que ejecutarlo explícitamente desde el constructor de éste último, si se estima conveniente.

PHP no soporta herencias múltiples, es decir una clase no puede ser hija de varias clases padre.

2. P.O.O.
Constructores y Destructores

Las clases que tienen un método constructor llaman a este método cada vez que se crea un nuevo objeto

```
void __construct ( [mixed $args [, $...]] )
```

El método destructor será llamado tan pronto como todas las referencias a un objeto en particular sean eliminadas o cuando el objeto sea explícitamente destruido .

```
void __destruct ( void )
```

2. P.O.O.
Visibilidad de propiedades/métodos

- **Public:** son accesibles desde cualquier punto del código

- **Protected:** sólo son accesibles por las clases heredadas (y por la propia clase que define el elemento)

- **Private:** sólo son accesibles por la clase que los definió

2. P.O.O.
Static / Operador de resolución

Declarar propiedades o métodos de clases como estáticos, los hace accesibles desde fuera del contexto del objeto.

```
public static $my_static = 'foo';
```

El operador de resolución permite llamar a un método o propiedad definido en una clase como estatico (*static*) sin necesidad de crear previamente un objeto.

```
clase :: metodo()
```

2. P.O.O.
Clases abstractas

Una clase abstracta es aquella de la que no se permite crear una instancia porque se ha definido como *abstract*.

Cualquier clase que contenga por lo menos un método abstracto debe también ser abstracta.

Los métodos definidos como abstractos simplemente declaran los métodos, no pueden definir la implementación.

```
abstract class AbstractClass
{
  abstract protected function getValue();
  abstract protected function prefixValue($prefix);
```

2. P.O.O. Interfaces

Las interfaces de objetos permiten crear código el cual especifica métodos que una clase debe implementar, sin tener que definir como serán manejados esos métodos.

Las interfaces son definidas usando la palabra reservada **interface**, de la misma manera que las clases estándar, pero sin que cualquiera de los métodos tenga su contenido definido.

```
interface iTemplate {
    public function setVariable($name, $var);
    public function getHtml($template);
}

class Template implements iTemplate
```

2. P.O.O. Final

- Evita que las clases hijo que extienden una clase puedan sobreescribir un método definido como **final**.
- Si la clase en sí misma es definida como **final** entonces no puede ser extendida.

2. P.O.O.
Funciones de manejo de objetos

PHP dispone de algunas funciones útiles a la hora de trabajar con clases y objetos:

`method_exists(obj, func) :` Comprueba si la función *func* existe en el objeto *obj*. Devuelve un valor booleano.

`property_exists(obj, prop) :` Comprueba si la propiedad *prop* existe en el objeto *obj*. Devuelve un valor booleano.

`get_class_vars(clase) :` Devuelve un array asociativo cuyos índices son los nombres de las variables y cuyos valores son los valores preasignados a esas variables.

`get_class_methods(clase):` Devuelve un array conteniendo los nombres de todos los métodos definidas en la clase *clase*

2. P.O.O.
Funciones de manejo de objetos

`get_object_vars(obj) :` Devuelve todas las variables y sus valores contenidas en el objeto *obj*.

`class_exists(clase) :` Devuelve un valor booleano indicando si la clase está definida.

`interface_exists(int) :` Devuelve un valor booleano indicando si el interface está definido.

3. Bases de Datos

3. Bases de Datos.
Conexión con una Base de Datos

Para poder utilizar las facilidades que proporciona un servidor de Bases de Datos, primero hay que establecer una conexión con el mismo.

En el caso de que el servidor sea MySQL, la conexión se establece con la función

```
mysql_connect(servidor, usuario, password)
```

Esta función devuelve un identificador de conexión o *False*, si la conexión no se puede establecer.

Extensión mejorada: mysqli_connect()

3. Bases de Datos.
Conexión con una Base de Datos

Los parámetros son todos opcionales, si no se especifican se intentaría una conexión con el servidor *localhost*, con nombre de usuario *root* y password en blanco

El parámetro servidor puede incluir un número de puerto. Si el puerto a utlizar es el 3306, no es necesario incluirlo en la llamada a *mysql_connect().*

3. Bases de Datos.
Conexión con una Base de Datos

Una vez ya no necesitemos una conexión, podremos liberarla con la función

```
mysql_close(conexion)
```

Dónde *conexion* es el identificador devuelto por una llamada a la función *mysql_connect()*

Si no se especifica ninguna conexión se cierra la última en haber sido abierta.

Las conexiones se liberan automáticamente al finalizar el script, aunque no se llame a esta función.

3. Bases de Datos.
Conexión con una Base de Datos

También podemos establecer una conexión persistente con el servidor con la función:

```
mysql_pconnect(servidor, usuario, password)
```

que funciona igual que *mysql_connect()*, con dos diferencias: antes de abrir la conexión verifica que no haya ninguna conexión persistente abierta y la conexión que abre no se cierra al finalizar el script, por lo que podemos propagar el identificador a la siguiente página. Esta práctica evita una sobrecarga de trabajo en el SGBD para abrir y cerrar conexiones

3. Bases de Datos. Operaciones DDL

Una vez conectados a un servidor podemos hacer básicamente tres cosas:

Crear una base de datos:

```
mysql_create_db(nombre, conexion)
```

Eliminar una base de datos:

```
mysql_drop_db(nombre, conexion)
```

Seleccionar una base de datos para consultarla:

```
mysql_select_db(nombre, conexion)
```

Extensión mejorada: mysqli_select_db()

3. Bases de Datos. Operaciones DDL

Veamos un ejemplo en el que aplicamos lo expuesto hasta ahora, para conectarse a un Servidor de BD, seleccionar una BD (llamada libros) y finalmente liberamos la conexión:

```
$c=mysql_connect("servidor", "root", "abcd");
mysql_select_db("libros", $c);
//  consulta a la BD libros y procesamiento
//  ........
mysql_close($c);
```

3. Bases de Datos.
Consultar una Base de Datos

La función *mysql_query()* envía una sentencia SQL al servidor para que este la ejecute. La sintaxis es:

```
mysql_query(consulta, conexion);
```

Esta función devuelve *True* o *False* según la consulta pueda o no ser ejecutada con éxito.

La consulta podrá ser cualquier consulta SQL válida, y por lo tanto podremos crear, modificar y eliminar tablas o índices, e insertar, borrar, actualizar o simplemente consultar los datos de la BD, siempre y cuando tengamos los permisos correspondientes.

Extensión mejorada: mysqli_query()

3. Bases de Datos.
Consultar una Base de Datos

Cuando la consulta es de tipo SELECT, *mysql_query()* devuelve un identificador de resultado, que otras funciones nos permiten manejar para procesar la información obtenida.

Algunas de ellas son:

```
mysql_result()
mysql_fetch_row()
mysql_fetch_array()
mysql_num_rows()
mysql_affected_rows()
```

3. Bases de Datos.
Consultar una Base de Datos

mysql_result() permite extraer un dato del resultado de una consulta, identificado por el número de fila y el número o nombre de columna.

A continuación mostramos su sintaxis y un ejemplo:

```
mysql_result(resultado, num_fila, num_o_nombre_col)

$resul=mysql_query("select * from libros", $c);
$titulo1=mysql_result($resul, 0, 2);
$titulo2=mysql_result($resul, 1, "titulo");
```

3. Bases de Datos.
Consultar una Base de Datos

mysql_fetch_row() permite extraer una fila completa del resultado de una consulta, almacenándola en un array, cada campo de la fila en un elemento del array. Devuelve falso si ya no hay más filas.

Su sintaxis es la siguiente:

```
mysql_fetch_row(resultado)
```

3. Bases de Datos.
Consultar una Base de Datos

PHP mantiene un puntero interno por cada resultado de una consulta, de forma que llamadas sucesivas a

mysql_fetch_row() devuelven filas consecutivas.

Esto nos permite escribir fácilmente bucles que recorran todos los datos de una consulta:

```
$resultado=mysql_query("select * from
libros", $c);
while($fila=mysql_row($resultado))
```

3. Bases de Datos.
Consultar una Base de Datos

El puntero interno de un resultado se puede mover a una posición determinada con la función:

```
mysql_data_seek(resultado,fila)
```

Extensión mejorada: mysqli_data_seek()

3. Bases de Datos.
Consultar una Base de Datos

mysql_fetch_array() es muy parecida a *mysql_fetch_row(),* pero devuelve un array asociativo, de manera que las claves son los nombres de los campos.

A continuación mostramos su sintaxis y un ejemplo:

```
mysql_fetch_array(resultado)

$resul=mysql_query("select * from libros", $c);
while($fila=mysql_fetch_array($resul))
  $titulo=$fila["titulo"]);
```

Extensión mejorada: mysqli_fetch_array()

3. Bases de Datos.
Consultar una Base de Datos

mysql_fetch_object() extrae una fila de un resultado como un objeto.

Este objeto tendrá tantas propiedades como atributos tiene el resultado, y con el mismo nombre. El valor de cada una de estas propiedades es el valor del correspondiente atributo en la fila extraida

La sintaxis es:

```
mysql_fetch_object(resultado)
```

Extensión mejorada: mysqli_fetch_object()

3. Bases de Datos.
Consultar una Base de Datos

mysql_num_rows() devuelve el número de filas contenidas en el resultado de una consulta SELECT.

mysql_affected_rows() devuelve el número de filas modificadas, insertadas o borradas en consultas de estos tipos.

La sintaxis de estas dos funciones son:

```
mysql_num_rows(resultado)
mysql_affected_rows(resultado)
```

Extensión mejorada: mysqli_num_rows()

3. Bases de Datos.
Consultar una Base de Datos

Cuando la función *mysql_query()* devuelve *False*, indicando que se ha producido un error, también genera un mensaje que aparecerá en la salida que ve el cliente, para controlar esta situación evitando que el usuario reciba el mensaje de error es habitual usar una estructura del tipo:

```
$respuesta=@mysql_query(consulta, conexion) or
   die (mensaje_al_usuario);
```

De esta manera el script acaba y al usuario le llega un mensaje escrito por nosotros.

3. Bases de Datos.
Consultar una Base de Datos

Dos funciones nos ayudan en el manejo y depuración de los errores:

`mysql_errno (conexión)`

Esta función nos da el número de error del último error mysql producido.

`mysql_error (conexión)`

Esta función da el mensaje correspondiente del último error mysql producido.

3. Bases de Datos.
Consultar una Base de Datos

Otras funciones que nos pueden ser útiles son:

```
mysql_field_flags()
mysql_field_len()
mysql_field_name()
mysql_field_table()
mysql_field_type()
mysql_fetch_lengths()
```

3. Bases de Datos.
Funciones para SQL Server

Veremos brevemente las funciones de las que PHP dispone para acceder a BDs SQL Server, la mayoría son muy similares a las ya vistas para MySQL.

La conexión al servidor se realiza con las funciones:

```
mssql_connect(servidor, usuario, password)
mssql_pconnect(servidor, usuario, password)
```

Para liberar la conexión:

```
mssql_close(conexion)
```

Seleccionar una Base de Datos

```
mssql_select_db(nombre_bd, conexion)
```

3. Bases de Datos. Funciones para SQL Server

Enviar una consulta al servidor:

```
mssql_query(consulta, conexion)
```

Para manejar el resultado de la consulta:

```
mssql_fetch_row(resultado)
mssql_fetch_array(resultado)
mssql_num_rows(resultado)
mssql_rows_affected(resultado)
```

3. Bases de Datos.
Funciones para ODBC

Además de las funciones vistas hasta el momento para realizar operaciones con bases de datos, es posible realizar operaciones con cualquier gestor de bases de datos comercial que soporte conexión mediante el protocolo ODBC. Para ello se pueden utilizar las funciones de las que dispone PHP 5 para la conexión mediante ODBC que se pueden encontrar en el Capítulo Funciones ODBC de la ayuda de PHP .

Las funciones disponibles tienen una sintaxis muy similar a las vistas para MySQL y MS SQL Server.

3. Bases de Datos. Funciones para ODBC

Algunas de las funciones más importantes son:

```
odbc_connect(dsn, usuario, password)
odbc_close(id de conexion)
odbc_errormsg()
odbc_exec(id de conexion, consulta)
odbc_fetch_row(id de resultado, fila)
```

Esta función localiza un registro en un resultado, si no se especifica el atributo *fila*, se accede a la siguiente fila.

```
odbc_result(id de resultado, columna)
```

Devuelve, de la fila localizada con *odbc_fetch_row(),* el campo indicado en *columna*, ya sea con el nombre del campo o el número de orden que ocupa. (Los índices comienzan en 1)

4. Sesiones y control de usuarios

4. Sesiones y control de usuarios
Las cookies

Las cookies son pequeños archivos de texto que el navegador guarda en el fichero del cliente.

La información allí guardada es accedida por el servidor Web que de esta manera puede personalizar la navegación para ese cliente, saber si es la primera vez que se visita el sitio desde ese ordenador, continuar la navegación en el último punto accedido, preferencias del cliente, etc.

4. Sesiones y control de usuarios
Las cookies

Cada cookie ocupa una línea de un fichero de texto, que puede contener hasta seis campos:

- Nombre de la cookie
- Valor, que es el contenido de la cookie
- Fecha de caducidad
- Dominio dentro del cual es válida la cookie
- Ruta
- Seguro, si se activa se establecerá una conexión segura para leer o escribir la cookie.

4. Sesiones y control de usuarios
Manejo de cookies desde PHP

La función *setcookie()* crea y envía al cliente una cookie. Sus argumentos son los seis elementos que puede tener una cookie:

```
setcookie (nombre, valor, caducidad, ruta, dominio, seguro)
```

De estos parámetros sólo es imprescindible el primero. Si no se pone el segundo, la cookie se borra del ordenador del cliente.

La función setcookie() *debe ejecutarse antes de enviar ninguna salida al navegador del cliente.*

4. Sesiones y control de usuarios
Manejo de cookies desde PHP

La caducidad se expresa en segundos de la era Unix, para que la cookie dure, por ejemplo, 5 minutos desde el momento de su creación podemos poner *time()+300.* Si queremos saltarnos un parámetro deberemos poner en el lugar correspondiente las comas y una cadena vacía (en el caso de valor, ruta y dominio) o 0 (si es el parámetro seguro)

4. Sesiones y control de usuarios
Manejo de cookies desde PHP

Ejemplos:

```
$usuario="Jorge";
setcookie("Nombre", $usuario);
```

Podemos establecer la caducidad en 10 minutos:

```
setcookie("Nombre", $usuario, time()+600)
```

En ambos casos se estará creando una variable *$_COOKIE[" Nombre”]*, que existirá mientras no caduque la cookie, y tendrá ámbito global.

4. Sesiones y control de usuarios
Manejo de cookies desde PHP

Las variables asociadas a cookies se guardan también en el array asociativo *$_COOKIE*, este array tiene como índices los nombres de las cookies.

Dependiendo de la configuración de PHP, también pueden estar en el array *$HTTP_COOKIE_VARS*.

Ejemplo:

```
echo "valor de la cookie: $_COOKIE["Nombre"];
```

4. Sesiones y control de usuarios
Manejo de cookies desde PHP

Se pueden asignar varios valores a una misma cookie dándole forma de array.

En el siguiente ejemplo creamos una cookie con tres valores, tres datos acerca de un libro.

```
setcookie("libro[0]", "El médico", time()+300);
setcookie("libro[1]", "Noah Gordon", time()+300);
setcookie("libro[2]", "1992", time()+300);
```

4. Sesiones y control de usuarios
Las cabeceras

Las cabeceras contienen diversa información que, de forma resumida, se puede decir que consiste en instrucciones al navegador de acciones que debe realizar (las cookies se envían como cabeceras) o de cómo debe mostrar la información que se le envía a continuación.

Las cabeceras debe ser lo primero que reciba el navegador, y de ahí que la función *setcookie()* deba ejecutarse antes de enviar cualquier otra salida al navegador.

4. Sesiones y control de usuarios
Las cabeceras

En PHP se puede enviar una cabecera al navegador con la función *header()*, su sintaxis es sencilla:

```
header (cadena_con_la_cabecera)
```

En HTML las cabeceras se introducen con la etiqueta *META*.

El contenido de la cadena que se pasa como parámetro a la función *header()* sería lo mismo que pondríamos en la etiqueta HTML después de la etiqueta *META*.

4. Sesiones y control de usuarios
Las cabeceras

Algunas cabeceras son las siguientes:

```
header("Content-Type: text/html");
```

que indica que lo que se envía a continuación es un documento de texto plano en HTML.

La siguiente, una cadena comenzada por "HTTP/1.0" transmite un código de estatus:

```
header("HTTP/1.0 401 Not Found");
header("Location: http://www.example.com");
```

Esta cabecera envía implícitamente, un codigo de estatus redirect.

4. Sesiones y control de usuarios
Las cabeceras

La cabecera expires indica cuando deja de ser válida la copia del documento albergada en la caché del cliente, si la fecha es ya pasada, se evita el almacenamiento en caché:

```
"Expires: Mon, 26 Jul 1997 05:00:00 GMT";
```

También se puede enviando estas otras cabeceras:

```
Last-Modified: ".gmdate("D, d M Y H:i:s")." GMT"
// HTTP/1.1
"Cache-Control: no-store, no-cache, must-revalidate";
"Cache-Control: post-check=0, pre-check=0", false;
// HTTP/1.0
"Pragma: no-cache";
```

4. Sesiones y control de usuarios
Control de usuarios

El control de usuarios tiene dos partes:

Identificar al usuario que se conecta y asegurarse de que es realmente quien dice ser (proceso conocido como autenticación)

Una segunda parte que consiste en personalizar la navegación para ese usuario, permitiendo que haga unas determinadas cosas y otras no, siguiendo su navegación para, por ejemplo, enviarle los objetos que haya ido añadiendo a su carrito de la compra, etc.

4. Sesiones y control de usuarios
Control de usuarios

La identificación de usuarios suele hacerse pidiéndole a éste que teclee una combinación de nombre de usuario y contraseña.

Esta autenticación puede hacerse mediante un script desde el propio servidor, pero también se puede pedir al navegador que realice esta tarea, enviándole una cabecera.

4. Sesiones y control de usuarios
Control de usuarios

La autenticación efectuada por el navegador tiene los siguientes pasos:

1. El servidor envía al navegador del cliente una cabecera de validación.
2. El navegador pide al usuario, en un cuadro de diálogo, el nombre de usuario y la contraseña.
3. El navegador devuelve al servidor estos datos y solicita cargar la página, cosa que sucede si hay coincidencia de los datos con los que tiene el servidor.

4. Sesiones y control de usuarios
Control de usuarios

Las cabeceras de validación se pueden enviar desde PHP con las instrucciones:

```
header ('WWW-Authenticate: Basic realm="Curso"');
header ('HTTP/1.0 401 Unauthorized');
```

El navegador al recibir esta cabecera, presenta un cuadro de diálogo solicitando el nombre de usuario y la contraseña.

Una vez leídos estos valores se situan en las variables de entorno:

```
$_SERVER['PHP_AUTH_USER']
$_SERVER['PHP_AUTH_PW']
```

4. Sesiones y control de usuarios
Sesiones

Una sesión es un bloque de información, guardado en el servidor, que almacena todo tipo de variables y valores que proveen información sobre los usuarios y sus visitas a nuestro sitio Web.

La sesión se creará en el momento que el usuario entre en el sitio Web.

Al crear la sesión se le asigna un identificador (Session ID o SID), en forma de cadena de caracteres, para asociarla al usuario.

4. Sesiones y control de usuarios
Sesiones

Este SID lo deberemos propagar de una a otra página cada vez que el usuario pincha un enlace. Así en la nueva página se podrá recuperar la sesión correspondiente.

La propagación puede hacerse de dos formas:

- Incluyéndolo en la URL, como un valor pasado por GET.
- Creando una cookie con el SID como valor.

4. Sesiones y control de usuarios
Abrir una sesión

Una sesión se arranca con la instrucción *session_start().*

Esta función devuelve siempre *True*.

Si llamamos a *session_start()* pasándole como parámetro un SID de una sesión ya abierta retomamos esa sesión.

Hay que tener en cuenta que si vamos a usar cookies para propagar el SID, tendremos que ejecutar esta función antes de enviar ninguna salida al navegador.

4. Sesiones y control de usuarios
Abrir una sesión

Para obtener el SID de una sesión recién abierta usaremos la función *session_id()*.

Cuando la sesión ya no nos sea útil podemos destruirla con la función *session_destroy()*.

4. Sesiones y control de usuarios
Poner nombre a una sesión

Llamando a la función *session_name()* sin ningún argumento recuperamos el nombre de la sesión, que por defecto es *PHPSESSID*.

Si la utilizamos antes de crear una sesión y ponemos una cadena como argumento ponemos ese nombre a la sesión.

En realidad eso significa que la cookie que se guarda en el ordenador del usuario tendrá ese nombre.

4. Sesiones y control de usuarios

Registrar información de una sesión

Para registrar cualquier información en nuestra sesión podemos hacerlo a través del array superglobal *$_SESSION* con la clave *'nombre'.*

```
$_SESSION['usuario']="Ana";
```

4. Sesiones y control de usuarios
Visión global

Por lo tanto un esquema general del trabajo con sesiones sería algo así:

El usuario entra en el sitio Web (identificándose o no), se crea la sesión y se guarda una cookie con su *SID*.

El usuario introduce alguna información en la página de entrada. Aquella información que nos interese para la sesión se registra en ella.

El usuario pincha un enlace que le lleva a otra página.

4. Sesiones y control de usuarios
Visión global

En la nueva página se recupera la cookie con el SID y se retoma la sesión llamando a *session_start()*

Parte de la información ya registrada en la sesión se utiliza para personalizar la página.

El cliente va recorriendo diversas páginas en las que se repite lo anterior y se va añadiendo nueva información a la sesión.

Finalmente, puede que parte de la información de la sesión se archive para un posterior uso, envío de productos solicitados por el cliente, etc.

4. Sesiones y control de usuarios
Más funciones para manejar sesiones

La función *session_unregister()* elimina una o más variables, cuyos nombres se pasan como parámetros, de la sesión actual.

La función *session_is_registered()* nos informa de si una variable está o no registrada en la sesión.

La función *session_encode()* devuelve una cadena con los valores de la sesión codificados.

La función *session_decode()* recupera los datos de una cadena codificada por *session_encode()*.

4. Sesiones y control de usuarios
Control de conexiones

Mientras el usuario está conectado a nuestro sitio web, nos puede interesar conocer y/o controlar las acciones del usuario en relación con esa conexión.

Internamente PHP reconoce tres estados de conexión:

0 – Conexión normal.

1 – Abortado, que se produce cuando el usuario detiene la carga de la página.

2 – Caducado que se produce si el usuario ha estado demasiado tiempo sin hacer nada con la sesión.

4. Sesiones y control de usuarios
Control de conexiones

La función *connection_status()* devuelve el estado de conexión en un campo de tres bits, cada bit corresponde a un estado según la lista anterior.

El tiempo máximo de ejecución de un script está fijado en *php.ini* en la directiva *max_execution_time*, pero podemos cambiar el valor para una página concreta en tiempo de ejecución con *set_time_limit()*.

Puede establecerse una función que se ejecutará cuando el script termine. Para ello se usa la función:

```
register_shutdown_function(funcion)
```

4. Sesiones y control de usuarios
Control de conexiones

La función de finalización se llamará tanto si el script acaba normalmente, es parado por el usuario o expira su tiempo de ejecución.

En esta función podemos usar las funciones *connection_status(), connection_timeout()* y *connection_aborted()* para comprobar si estas han sido las causas de la finalización del script.

En todo caso no podremos enviar ninguna salida al navegador desde esta función.

4. Sesiones y control de usuarios
Control de conexiones

Con la función *ignore_user_abort()* hacemos que el script continúe ejecutándose aunque el usuario detenga la carga de la página. El usuario deja de recibir el resultado de la ejecución, pero ésta continua hasta el final del script.

4. Sesiones y control de usuarios
Control de conexiones

Otras funciones de control de conexión son:

exit() que nos permite terminar la ejecución de un script, y consecuentemente, la salida hacia el usuario.

die() que funciona como *exit()*, pero permite incluir un mensaje que se envía al navegador del cliente.

sleep() retrasa la ejecución de un script tantos segundos como se le pase como argumento.

usleep() es igual que la anterior, pero el valor pasado es interpretado como microsegundos.

5. Ficheros PDF

5. Ficheros PDF
Configuración previa

Para poder utilizar las funciones que desde PHP nos permiten crear ficheros PDF, debemos asegurarnos de que la línea

```
extension=php_pdf.dll
```

está activa en el fichero *php.ini*

5. Ficheros PDF
Unidades de medida

Las unidades de medida que se manejan son los puntos, que equivalen a 1/72 pulgadas o, aproximadamente, 0,35 mm.

1 cm equivale a unos 28,5 puntos.

Las medidas de un A4 en puntos son:

Altura 842 puntos.

Anchura 595 puntos.

5. Ficheros PDF
Creación de ficheros PDF

Lo primero que hay que hacer es crear un PDF mediante *pdf_new()*.

A continuación asociamos el PDF a un archivo con la función *pdf_open_fila()* a la que pasamos el identificador de PDF obtenido antes, y el nombre del fichero.

```
$pdf=pdf_new();
pdf_open_file($pdf, "nombre.pdf");
```

Una vez acabamos de trabajar con el fichero lo cerraremos con *pdf_close()*.

5. Ficheros PDF
Páginas

Podemos insertar una página en un documento PDF con la orden *pdf_begin_page()*.

```
pdf_begin_page($g, x, y);
```

$g es el descriptor del fichero PDF, *x* e *y* son la altura y la anchura de la página. Por tanto cada vez que iniciamos una nueva página, sus dimensiones son independientes del resto del documento.

Al finalizar la página llamaremos a *pdf_end_page()*. Es preciso no poner un comienzo de página sin haber finalizado la anterior.

5. Ficheros PDF
Tipos de letra

En el directorio *pdf-related* de la instalación de PHP, hay un documento llamado *pdflib.upr*, dónde se encuentra un listado con los nombres de las fuentes que podremos utilizar.

Para definir un tipo de letra a utilizar dentro de un documento PDF se usan las funciones *pdf_load_font()* y *pdf_setfont()*. Una vez definida una fuente se usará en toda la página hasta que no se defina otra distinta.

5. Ficheros PDF
Tipos de letra

La sintaxis de *pdf_load_font()* es la siguiente:

`pdf_load_font(doc, letra, code, “”)`

doc es el identificador del documento PDF

letra es el nombre de un tipo de letra, debe ir entre comillas y coincidir con uno de los tipos definidos en *pdflib.upr*

code es el tipo de codificación. Usaremos el valor *"host“*

La sintaxis de *pdf_setfont()* es la siguiente:

`pdf_setfont(doc, fuente, tam)`

doc es el identificador del documento PDF

fuente es el identificador de la fuente obtenida de la función anterior

tam es el nº de pixels de la fuente

5. Ficheros PDF
Insertar texto

La función *pdf_show()* inserta texto en el documento utilizando la última fuente definida. Esta función no tiene en cuenta si el texto cabe o no en la línea y si se hacen dos llamadas consecutivas se continuará en la misma línea.

La función *pdf_continue_text()* escribe un texto en la línea siguiente.

La sintaxis de ambas funciones es la misma:

```
pdf_show(documento, texto)
pdf_continue_text(documento, texto)
```

5. Ficheros PDF
Situando el origen de coordenadas

Las coordenadas son cartesianas, el origen esta en la esquina inferior izquierda y crecen hacia arriba y hacia la derecha.

Se puede situar el origen en otro punto de la página con la función *pdf_translate().* Su sintaxis es:

```
pdf_translate(documento, x, y)
```

5. Ficheros PDF
Escribiendo texto posicionado

La función *pdf_show_xy()*, escribe un texto a partir de las coordenadas que se le indican. Hay que tener en cuenta que el texto se situará hacia arriba y la derecha del punto dado por las coordenadas.

La sintaxis es:

```
pdf_show_xy(documento, texto, x, y)
```

5. Ficheros PDF
Cajas de texto

Una caja de texto es un bloque de un determinado tamaño en el que se ajusta un texto con saltos de línea automáticos, y con la alineación horizontal que se especifique. Para escribir un texto en una caja se usa la función *pdf_show_boxed()*, cuya sintaxis es:

```
pdf_show_boxed(documento, texto, x, y, a, h, alin)
```

Los posibles valores para *alin* son: *"left"*, *"right"*, *"center"* y *"justify"*.

Sólo se escribe el texto que cabe en la caja, la función devuelve cuantos caracteres han quedado sin mostrar.

5. Ficheros PDF
Establecer el interlineado

La función *pdf_set_value()* permite establecer varios parámetros del documento. Su sintaxis es:

```
pdf_set_value(documento, nombre, valor)
```

El argumento *nombre* indica que parámetro del documento cambia la función. Si queremos establecer el interlineado deberemos poner *"leading"* en esa posición. El valor que pasamos en el tercer argumento corresponderá al interlineado en puntos.

5. Ficheros PDF
Determinar la anchura de un texto

Para averiguar la anchura de un texto podremos usar la función *pdf_stringwidth()* que nos da el valor expresado en puntos, para la fuente seleccionada en un momento dado. Su sintaxis es:

```
pdf_stringwidth(documento, texto)
```

Con esta información podremos dimensionar cajas de texto, saber si nos va a caber en una línea, etc.

5. Ficheros PDF
Trabajar con escalas

Se puede cambiar la escala a la que se muestran los textos y otros objetos en el documento tanto en vertical como en horizontal. Una escala 2 en vertical significa que las letras saldran el doble de altas.

La función que establece la escala es *pdf_scale()* cuya sintaxis es:

```
pdf_scale (documento, escala_h, escala_v)
```

La escala que se establece es con respecto a la anteriormente vigente. Para neutralizar una escala 2 en vertical habría que establecer la escala a 0,5.

5. Ficheros PDF
Trabajar con escalas

La escala tambien afecta a las medidas, por lo que con una escala de 2 en horizontal, la medida a lo ancho de la página será la mitad, y si mandamos mostrar un texto a partir de la coordenada 100, aparecerá a partir de lo que normalmente sería el punto de coordenada 200.

Si los valores de escala son negativos se produce una simetría en el correspondiente sentido. Con una escala horizontal de –1, se escribiría de derecha a izquierda, y las coordenadas de la página serían negativas, disminuyendo hacia la derecha.

5. Ficheros PDF
Rotaciones

Para hacer que un texto (o figuras) aparezcan rotadas en un cierto ángulo se usa la función *pdf_rotate()*. Su sintaxis es:

```
pdf_rotate(documento, angulo)
```

Si los valores de escala son negativos se produce una simetría en el correspondiente sentido. Con una escala horizontal de –1, se escribiría de derecha a izquierda, y las coordenadas de la página serían negativas, disminuyendo hacia la derecha.

5. Ficheros PDF
Bloques intermedios

Para facilitar el cambio de escalas, rotaciones, etc. en un bloque de texto y luego recuperar las condiciones anteriores a este bloque PHP dispone de dos funciones. *pdf_save()* y *pdf_restore()*, la primera guarda el entorno y la segunda vuelve a establecer el entorno anteriormente salvado. La sintaxis es igual en ambos casos:

```
pdf_save(documento)
pdf_restore(documento)
```

5. Ficheros PDF
Efectos en el texto

Las siguientes funciones permiten una serie de efectos en el texto que se muestra en un PDF:

```
pdf_set_parameter(documento, efecto, valor)
```

Donde *efecto* puede tomar los valores *"underline"*, *"overline"* o *"strikeout"*, que significan subrayado, linea por encima del texto o tachado. El tercer parámetro puede contener *"true"* o *"false"* para activar o desactivar el efecto.

Esta función no respeta las excepciones creadas con *pdf_save()* y *pdf_restore().*

5. Ficheros PDF
Efectos en el texto

La funcion *pdf_set_value(),* ya vista, permite varios efectos.

Con *"textrendering"* como segundo argumento, el tercer valor tiene los siguientes significados:

0: texto normal, desactiva las opciones establecidas.

1: sólo se muestra el contorno de la letra.

2: dibuja el contorno de un color y el relleno de otro.

3: texto invisible.

5. Ficheros PDF
Cambiar el color del texto

El color se establece con la función *pdf_setcolor():*

```
pdf_setcolor(documento, zona, "rgb", r, g, b, null)
pdf_setcolor(documento, zona, "cmyk", c, m, y, k)
```

El parámetro zona será *"fill"* para referirse al relleno de toda la letra o *"stroke"* para el contorno.

El color de relleno afecta automáticamente a todas las funciones que muestran texto, pero el color de contorno precisa de una llamada a *pdf_set_value().*

Como se ve se puede establecer el color utilizando el modelo RGB o el CMYK, tambien se puede usar una escala de grises, con la palabra *"grey"*.

5. Ficheros PDF
Imágenes en documentos PDF

Se pueden insertar imágenes de formatos *jpg*, *gif*, *png* y *tiff*.

Para insertar la imagen tenemos que obtener un identificador de imagen:

```
$v=pdf_load_image(documento, tipo, imagen)
```

El parámetro imagen es el nombre del archivo con extensión y ruta, si es preciso. y el parámetro tipo es *“gif”*, *“jpeg”*, *“tiff”* o *“png”.*

5. Ficheros PDF
Imágenes en documentos PDF

Obtenido el identificador de imagen ya podemos insertarla en el documento:

```
$v=pdf_place_image(documento, imagen_id, x, y, escala)
```

Una vez insertada la imagen liberamos recursos, eliminando de memoria la imagen mediante la función:

```
pdf_close_image(documento, imagen_id)
```

5. Ficheros PDF
Imágenes en documentos PDF

Podemos saber la anchura y altura de una imagen una vez abierta, de este modo podremos saber el factor de escala a aplicar si debemos adaptarla a un determinado espacio. Esta información se obtiene con la función *pdf_get_value()*:

```
pdf_get_value(documento, "imagewidth", imagen_id)
pdf_get_value(documento, "imageheight", imagen_id)
```

Esta función es el reverso de la ya vista *pdf_set_value()* y sirve para obtener varios valores numéricos, dependiendo del segundo argumento.

5. Ficheros PDF
Gráficos en documentos PDF

PHP permite insertar elementos como segmentos, líneas poligonales, arcos, etc.

Para dibujar un segmento utilizaremos *pdf_moveto()* para fijar el punto de partida y luego *pdf_lineto()* para trazar el segmento. Luego hay que dar una orden para que se dibuje la línea con *pdf_stroke()*.

```
pdf_moveto(documento, x, y)
pdf_lineto(documento, x, y)
pdf_stroke(documento)
```

5. Ficheros PDF
Gráficos en documentos PDF

Podemos cambiar colores, espesores y tipos de líneas con las siguientes funciones:

```
pdf_setcolor(documento, “stroke”, r, g, b, null)
pdf_setcolor(documento, “fill”, r, g, b, null)
pdf_setlinewidth(documento, ancho)
pdf_setdash(documento, t, b)
```

Esta última establece el tipo de línea, continua, de puntos o de trazos, *t* es la longitud en puntos de cada trazo y *b* el número de puntos que se dejan en blanco entre trazos. Si *b* es 0 la línea es continua.

5. Ficheros PDF
Gráficos en documentos PDF

Para trazar una poligonal insertaremos varios *pdf_lineto()* antes de *pdf_stroke()*.

Podemos cerrar la línea poligonal automáticamente con *pdf_closepath()*.

Si lo hacemos así aparecerá sólo la línea, si sustituimos *pdf_stroke()* por *pdf_fill()* se rellena la figura, creándose un recinto cerrado aunque la línea poligonal fuese abierta.

pdf_fill_stroke() rellena la figura y muestra también el contorno, si la línea fuese abierta no aparecería en el contorno la línea de cierre.

5. Ficheros PDF
Gráficos en documentos PDF

Trazamos un rectángulo con:

```
pdf_rect(documento, x, y, ancho, alto)
```

La función que traza un círculo es:

```
pdf_circle(documento, x, y, r)
```

Un arco de circunferencia ser traza con:

```
pdf_arc(documento, x, y, r, ang_ini, ang_fin)
```

Parta trazar una elipse o un arco de elipse se ha de utilizar las dos anteriores deformando el círculo con un cambio de escala.

5. Ficheros PDF
Visualización de PDF

En lugar de crear un documento PDF, podemos abrir un documento ya existente y enviarlo al navegador.

Bastará mandar dos cabeceras:

```
header("Content-type: application/pdf");
header("Content-length: valor");
```

Para saber la longitud de la información podemos usar previamente la función *filesize().*

Por ultimo enviaremos el contenido del fichero:

```
readfile(nombre_fichero);
```

5. Ficheros PDF
Visualización de PDF

Otra posibilidad es crear documentos PDF sólo en memoria, para enviarlos al navegador y visualizarlos en él.

Para ello deberemos usar la función *pdf_new()* para crear un nuevo documento. A continuación usaremos *pdf_open_file()* para abrir el documento.

Ejemplo:

```
$p=pdf_new();
pdf_open_file($p);
```

5. Ficheros PDF
Visualización de PDF

A continuación rellenaríamos el documento con las funciones ya vistas, utilizando el descriptor devuelto por *pdf_open_file()* , en nuestro caso *$p*.

Al final usaremos *pdf_get_buffer()* para obtener el contenido el documento, obtenemos su longitud, enviamos las cabeceras y, finalmente, enviamos el contenido del documento.

```
$buffer=pdf_get_buffer($p);
$long=strlen($buffer);
header("Content-type: application/pdf");
header("Content-length: $long");
echo $buffer;
```

6. Gráficos e imágenes

6. Gráficos e imágenes
Tratamiento de imágenes

PHP permite leer y enviar al navegador ficheros de imágenes de diversos formatos, pero también crear imágenes dinámicas en memoria, dibujar elementos gráficos en ellas y luego guardarlas como ficheros y/o enviarlas al navegador.

Para que esto sea posible, es necesario compilar PHP con la biblioteca de funciones de imágenes *GD Graphics Library*. El manejo de imágenes dinámicas requiere que esté instalada la librería de PHP llamada *php_gd2.dll*. En la versión de PHP que estamos manejando se instala por defecto pero requiere que esté activada en *php.ini*.

6. Gráficos e imágenes
Tratamiento de imágenes

Si un script envía una imagen al navegador deberá ir encabezado por el header:

```
header("Content-type: image/jpeg");
```

obviamente después de la / se pondrá el formato de imagen que se envía: jpeg, gif, png, bmp.

6. Gráficos e imágenes
Creación de una imagen

Se crea una imagen dinámica con la función *imagecreate()*, cuya sintaxis es:

```
imagecreate(x, y);
```

x e *y* son el ancho y el alto de la imagen. Cuando se crea la imagen no se especifica un formato, pues este sólo es necesario especificarlo a la hora de enviarla al navegador o guardarla en un archivo.

La función devuelve un descriptor de objeto, que será el que utilicemos en lo sucesivo para manejar la imagen.

6. Gráficos e imágenes
Funciones para elaborar imágenes

Las siguientes funciones dibujan diferentes figuras:

```
imageline(imagen, x1, y1, x2, y2, color);
imagedashedline(imagen, x1, y1, x2, y2, color);
imagerectangle(imagen, x1, y1, x2, y2, color);
imagearc(imagen, x, y, anch, alt, ini, fin, color);
imageellipse(imagen, x, y, anchura, altura, color);
imagepolygon(imagen, vertices, num_ver, color);
```

Las versiones rellenas de estas figuras, se distinguen por la palabra *filled* justo despues de image, p.e.: *imagefilledrectangle()*. El arco relleno, es un poco especial pues tiene un parámetro adicional que define el estilo.

6. Gráficos e imágenes
Funciones para elaborar imágenes

La función *imagefilledtoborder()* rellena de color desde el punto definido hasta que se encuentra un borde del color que se especifica:

```
imagefilledtoborder(img, x, y, colorborde, color);
```

La función *imagecolorat()* devuelve el color del punto que se especifica.

```
imagecolorat(imagen, x, y);
```

6. Gráficos e imágenes
Funciones para elaborar imágenes

La función:

```
imagettftext(img, tam, ang, x, y, col, fnte, txt);
```

Escribe un texto en la imagen *img*, se indica el tamaño de fuente, el ángulo de rotación, el punto de inicio, el color, la fuente true type (un fichero .ttf), y el texto a escribir.

```
imagestring(img, font, x, y, txt, col)
```

Al igual que la anterior, escribe un texto, pero con una fuente interna, seleccionada en el parámetro *font*, con un valor de 1 a 5.

6. Gráficos e imágenes
Mostrar imágenes

La familia de funciones *imagexxx*(), envia al navegador una imagen. El formato con que se envía depende del sufijo:

```
imagegif(imagen);
imagejpeg(imagen);
imagepng(imagen);
imagewbmp(imagen);
```

Nota: se pueden enviar imágenes *jpeg* entrelazadas, si se activa la opción antes del envío con *imageinterlace()*.

6. Gráficos e imágenes
Imágenes y archivos

Las funciones anteriores admiten como segundo parámetro un nombre de archivo. En este caso, no se envía la imagen al navegador, sino que se guardan en el fichero especificado.

La familia de funciones *imagecreatefromxxx()* abren un fichero de imagen y la cargan en memoria como una imagen dinámica cuyo descriptor devuelve.

```
imagecreatefromxxx(fichero);
```

Los sufijos que se pueden poner son los ya vistos.

6. Gráficos e imágenes
Otras funciones de imágenes

getimagesize() devuelve una matriz de cuatro elementos: anchura, altura, tipo (1=gif, 2=jpeg, 3=png) y una cadena de formato "heigth=xxx width=xxx".

```
getimagesize(fichero);
```

Debemos destruir las imágenes, para liberar recursos, una vez que ya no las vamos a usar mas, la función *imagedestroy()*, sirve para esto:

```
imagedestroy(imagen);
```

6. Gráficos e imágenes
Otras funciones de imágenes

imagecopyresized() crea una copia y redimensiona parte de una imagen, su sintaxis es:

```
imagecopyresized(img_dest, img_org, x_dest, y_dest,
x_org, y_org, anch_dest, alt_dest, anch_org,
alt_org);
```

Una función similar es *imagecopyresampled()* pero además de la copia realiza un suavizado por interpolación que mejora la calidad. Los argumentos de esta función son los mismos.

7. Utilidades varias

7. Utilidades varias
Uso de objetos COM desde PHP

Las funciones COM son una vía que permite identificar e intercambiar información entre objetos y componentes de diferentes sistemas, arquitecturas, lenguajes y máquinas. COM es el modelo más usado para conectar software de diferentes fabricantes.

Proporciona un amplio conjunto de herramientas cliente-servidor de servicios integrados, fáciles de usar y eficientes en la conexión de los componentes de variados lenguajes.

7. Utilidades varias
Uso de objetos COM desde PHP

Veremos con ejemplos de Word y Excel como se trabaja con este modelo.

Lógicamente estos programas deben estar instalados en la máquina.

Dentro de PHP, COM se comporta como una clase de la que es preciso siempre hacer al menos una instancia creando un objeto. Así pues, para poder usar las funciones que vamos a explicar a continuación, es necesario crear un objeto con la sentencia *new COM*.

7. Utilidades varias
Uso de objetos COM desde PHP

Para acceder desde PHP a MS Word, usamos esta instrucción para crear el objeto:

```
$word = new COM("word.application")
        or die("No se puede instanciar el Word");
```

El parámetro indica el módulo al que vamos a acceder. Opcionalmente, pueden indicarse también el nombre del servidor desde el cual será instanciada la clase y el código de página que debe utilizarse para hacer compatibles las cadenas entre las dos aplicaciones.

7. Utilidades varias
Uso de objetos COM desde PHP

Una vez creado el objeto ya podemos hacer referencia al mismo e indicarle que haga cosas. Los métodos y propiedades de cada objeto COM dependen del propio objeto, en el caso de Word podríamos hacer algo como lo siguiente:

```
$word->Visible = 1;
$word->Documents->Add();
$word->Selection->Font->size=16;
$word->Selection->Font->bold=true;
$word->Selection->TypeText("Esto es una prueba.");
$word->Documents[1]->SaveAs($nombre_fichero);
```

7. Utilidades varias
Funciones criptográficas

PHP utiliza la librería *mcrypt* para encriptar un texto. Esta librería soporta una gran variedad de algoritmos de bloque como DES, TripleDES, Blowfish (por defecto), 3-WAY, SAFER-SK64, SAFER-SK128, TWOFISH, TEA, RC2 y GOST en los modos de cifrado CBC, OFB, CFB y ECB. Adicionalmente, soporta RC6 e IDEA, que se consideran "no-libres".

Cada uno de estos algoritmos se identifica por un valor numérico. Una serie de constantes de la forma *MCRYPT_XXX* facilitan su uso.

7. Utilidades varias
Funciones criptográficas

Mcrypt puede operar en 6 modos de cifrado:

ECB, que es adecuado para encriptar datos aleatorios y cadenas no muy largas.

CBC, que es muy adecuado para encriptar ficheros.

CFB, que es la forma más adecuada de encriptar paquetes de bytes uno a uno.

OFB, que es similar al modo CFB, si bien es más adecuado para encriptar el código de aplicaciones informáticas, ya que en éstas la aparición de errores debe evitarse.

NOFB, como OFB pero en n-bits

Stream, para WAKE y RC4

7. Utilidades varias
Funciones criptográficas

Para realizar un cifrado hay que dar los siguientes pasos:

Abrir el módulo del algoritmo que vamos a utilizar e indicar el modo de encriptación, con la función:

```
mcrypt_module_open(alg, dir_alg, modo, dir_modo)
```

Esta función devuelve un identificador de encriptación o *False* si se ha producido algún error.

7. Utilidades varias
Funciones criptográficas

La explicación de los parámetros de *mcrypt_module_open()* es la siguiente:

alg es la constante que identifica el algoritmo de encriptación a utilizar.

dir_alg es el directorio del algoritmo, normalmente se deja por defecto poniendo "".

7. Utilidades varias
Funciones criptográficas

modo es la constante que identifica el modo de cifrado, los valores que puede tomar son:

MCRYPT_MODE_ECB

MCRYPT_MODE_CBC

MCRYPT_MODE_CFB

MCRYPT_MODE_OFB

MCRYPT_MODE_NOFB

MCRYPT_MODE_STREAM

dir_modo es el directorio del modo, habitualmente se deja por defecto poniendo "".

7. Utilidades varias
Funciones criptográficas

Seguidamente, hay que crear un vector de inicialización (*IV*) que sirve de semilla para iniciar el proceso de encriptación, con la función:

```
mcrypt_create_iv(tamaño, valor_inicial)
```

Para hallar el tamaño del vector se usa la función:

```
mcrypt_enc_get_iv_size(td)
```

El parámetro *td* es el identificador devuelto por *mcrypt_module_open()*.

7. Utilidades varias
Funciones criptográficas

En *valor_inicial* lo normal es indicar, poniendo *MCRYPT_RAND* , que la función utilice un número aleatorio.

Con todo lo visto, la utilización de *mcrypt_create_iv()* suele quedar como se muestra a continuación, con una instrucción antes para inicializar el generador de números aleatorios (no es neceario en versiones recientes):

```
srand((double)microtime()*1000000);
mcrypt_create_iv(mcrypt_enc_get_iv_size($td),MCRYPT
_RAND);
```

7. Utilidades varias
Funciones criptográficas

El tercer paso es inicializar los *buffers* necesarios y establecer la clave para la encriptación:

```
mcrypt_generic_init(td, clave, iv);
```

La encriptación de los datos es, quizá, el paso más sencillo:

```
mcrypt_generic(td, texto);
```

Finalmente, liberamos todos los *buffers* y cerramos todos los módulos usados:

```
mcrypt_generic_deinit(td);
```

7. Utilidades varias
Funciones criptográficas

Para desencriptar texto se usa la función *mdecrypt_generic():*

```
mdecrypt_generic(td, texto);
```

7. Utilidades varias
Funciones de interprete XML

Comencemos explicando brevemente que es XML. XML no es un lenguaje definido, sino unas normas sintácticas para crear lenguajes de etiquetas.

El usuario define las etiquetas que va a usar. Lo que signifique cada etiqueta depende de la utilidad que el usuario necesite.

Así pues para crear un interprete XML, lo primero que hay que definir son las etiquetas que se van a usar.

7. Utilidades varias
Funciones de interprete XML

En PHP un analizador XML o parser se crea con la función *xml_parser_create()*, que no tiene parámetros y devuelve un identificador.

El parser creado admite un texto codificado en XML, e irá troceándolo, identificando las etiquetas de apertura, de cierre y el texto contenido entre ellas.

Cada vez que el parser identifica una etiqueta de apertura, llamará a una función definida a tal fin por el usuario. Para las etiquetas de cierre y el contenido ocurre lo mismo con otras dos funciones de usuario.

7. Utilidades varias
Funciones de interprete XML

Para decirle al parser cuales son las tres funciones de usuario a las que debe llamar se utilizan las llamadas:

```
xml_set_element_handler(parser, f_ini, f_final);
xml_set_character_data_handler(parser, f_datos);
```

parser es el identificador de analizador

f_ini es la función que debe llamar el parser para procesar las etiquetas de apertura, *f_final* es la función para la etiqueta de cierre.

f_datos es la función que procesará el contenido entre dos etiquetas.

7. Utilidades varias
Funciones de interprete XML

Para evitar problemas con mayúsculas y minúsculas se suele forzar al parser a pasar a mayúsculas todas las etiquetas. Esto se hace llamando a la función:

```
xml_parser_set_option(parser, opcion, valor);
```

Para establecer las mayúsculas opción debe valer *XML_OPTION_CASE_FOLDING y* valor debe valer *True*.

Luego deberemos tener en cuenta que las etiquetas debemos manejarlas siempre en mayúsculas.

7. Utilidades varias
Funciones de interprete XML

Veamos ahora como deben ser las funciones de usuario.

La función que procesa las etiquetas de apertura tiene tres argumentos.

El primero es el identificador del parser, el segundo es la etiqueta que provocó la llamada, y el tercero es un array asociativo con los atributos de la etiqueta organizados de la forma *nombre_atributo=>valor*.

7. Utilidades varias
Funciones de interprete XML

La función que procesa las etiquetas de cierre tiene dos argumentos.

El primero es el identificador del parser y el segundo es la etiqueta que provocó la llamada.

Finalmente la función que procesa los datos tiene también dos argumentos, el identificador y el texto entre las etiquetas.

7. Utilidades varias
Funciones de interprete XML

Una forma práctica de manejar las etiquetas es crear dos arrays asociativos, uno para las etiquetas de apertura y otro para las etiquetas de cierre.

Las etiquetas serán las claves del array y en el valor correspondiente pondremos un texto que se deba mostrar en el navegador, o utilizar de alguna otra forma cada vez que aparezca la correspondiente etiqueta.

7. Utilidades varias
Funciones de interprete XML

Una vez definidos todos estos elementos podemos ya utilizar el parser, suministrándole datos para que los analice, llamando a la función:

```
xml_parse(parser, datos, es_final)
```

Los datos a analizar van en el parámetro *datos.* El parámetro es_final es opcional y si existe y es True significa que este bloque de datos es el último que hay que analizar.

7. Utilidades varias
Funciones de interprete XML

Cuando ya no se vaya a usar más el parser deberemos liberar recursos con la función *xml_parser_free()* que tiene como argumento el identificador del parser que queremos liberar.

7. Utilidades varias

Acceso a directorios activos de W2000

No hay mucha información sobre la utilización del Directorio Activo de MS Windows 2000 Server y 2003 Server, pero dado que se trata de un directorio ligero, podemos acceder a el haciendo uso de las funciones LDAP (Light Directory Access Protocol) de PHP.

7. Utilidades varias
Acceso a directorios activos de W2000

Las funciones más importantes para acceso a LDAP son:

`ldap_connect(nombre_host, puerto)`

Conecta a un servidor ldap, devuelve un identificador de conexión. El segundo argumento es opcional.

`ldap_bind(conexion, usuario, contraseña)`

Autentifica a un usuario contra un servidor LDAP.

7. Utilidades varias
Acceso a directorios activos de W2000

`ldap_search(conex, dn_base, filtro, atrs)`

Realiza una búsqueda en un directorio LDAP, devolviendo un identificador de resultado.

dn_base es la unidad organizativa de Active Directory

filtro establece las condiciones de búsqueda

atrs es un array en el que se indican que atributos queremos recibir como resultado de la búsqueda. Por razones de eficiencia es muy recomendable limitar los atributos devueltos.

7. Utilidades varias
Acceso a directorios activos de W2000

`ldap_get_entries(conexion, resultado)`

Obtiene, a partir de un resultado de búsqueda, un array multidimensional en el que están todos los datos del resultado. La estructura del array devuelto *v* es la siguiente:

v["count"]: número de entradas del resultado.

v[i]["dn"]: *dn* de la entrada i-ésima

v[i]["count"]: número de atributos de la entrada i-ésima

v[i][j]: j-ésimo atributo de la entrada i-ésima

v[i]["atr"]["count"]: numero de valores del atributo *atr* de la entrada i-ésima

v[i]["atr"]["count"]: valor j-ésimo para *atr* de la entrada i-ésima

7. Utilidades varias
Excepciones

El control de errores en PHP5 se realiza mediante excepciones.

Para lanzar una excepción se usa **thrown new Excepcion(mensaje_error)**

Las excepciones producidas en un bloque **try** se capturan en un bloque **catch (Exception e)**

```
try {
}
catch (Exception $e) {
echo $e->getMessage();
}
```

7. Utilidades varias
Excepciones

Se pueden personalizar las Excepciones extendiendo la clase Exception y redefiniendo el contructor y el método __toString().

```
class MyException extends Exception{
public function __construct($message, $code = 0) {
        // codigo
        parent::__construct($message, $code);
}
public function __toString() {
  return $this->code. $this->message;
}
```