# Introducción a PHP

## (UD 1)

# 1. Desarrollo web del lado del servidor

- Cada vez es más habitual ejecutar programas remotos a través del navegador de internet
  - Solución **CLIENTE - SERVIDOR**.
- El navegador de internet es el programa CLIENTE, que lanza peticiones al SERVIDOR.
- El SERVIDOR es otra máquina remota, en la que corren los programas SERVIDORES (p.ej: el servidor http es el que nos sirve las páginas web para poder verlas en el cliente)
- Desde la web se puede así acceder a los recursos del servidor. Por ejemplo, mediante **instrucciones SQL** puede usarse una BD alojada en el servidor.

# 1. Desarrollo web del lado del servidor

- **Esquema CLIENTE – SERVIDOR**

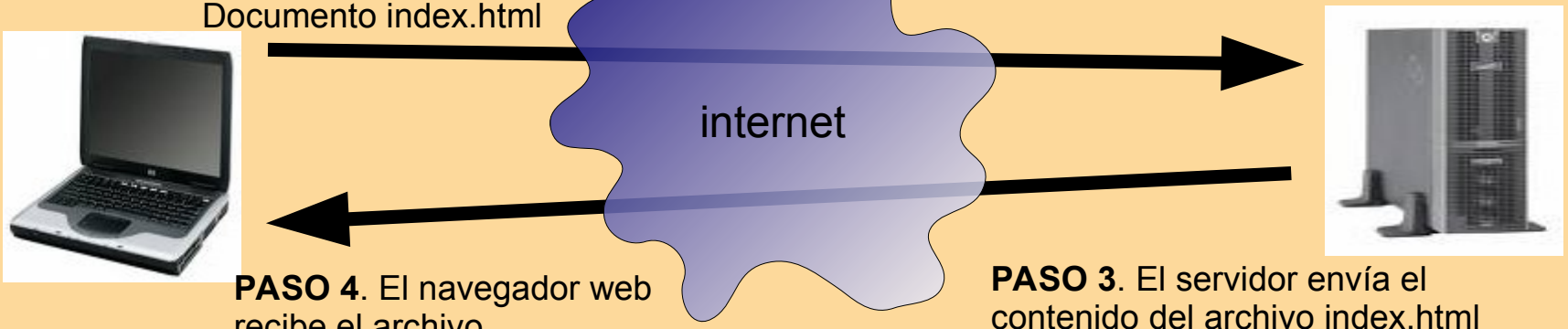
  Ejemplo: servicio www

**Máquina CLIENTE.**
Ejecuta el programa cliente (navegador web)

**Máquina SERVIDOR.**
IP: 210.38.1.34.
Ejecuta el programa servidor (servidor web)

**PASO 1**. Solicitud de página a la máquina 210.39.1.34. Documento index.html

**PASO 2**. El programa servidor captura la petición y busca la página en el directorio adecuado




**PASO 3**. El servidor envía el contenido del archivo index.html

**PASO 4**. El navegador web recibe el archivo index.html, lo interpreta y lo muestra por pantalla.

# 1. Desarrollo web del lado del servidor

- **Esquema CLIENTE – SERVIDOR**

  Otro ejemplo de servicio www

**Máquina CLIENTE.**
Ejecuta el programa cliente (navegador web)

**Máquina SERVIDOR.**
IP: 210.38.1.34.
Ejecuta el programa servidor (servidor web)

**PASO 1**. Solicitud de página a la máquina 210.39.1.34. Documento consulta.php

**PASO 2**. El programa servidor busca la página y descubre que se trata de un script PHP.

internet

**PASO 3**. El servidor ejecuta el script (que probablemente contiene código SQL para acceder a alguna base de datos) y envía el resultado, que debe ser código HTML.

**PASO 4**. El navegador web recibe el código html, lo interpreta y lo muestra por pantalla.

# 1. Desarrollo web del lado del servidor

- **DHTML y PHP** son los lenguajes que nos van a permitir ejecutar programas en el servidor y acceder a sus recursos a través de páginas web.
  - Existen otras posibilidades, como:
    - DHTML con ASP
    - DHTML con JSP
    - DHTML con Python, Ruby, Perl, etc.
- **MySQL / MariaDB** es un servidor de bases de datos que nos permitirá conectarnos a ellas y ejecutar sentencias SQL de forma remota al visitar una página web.
  - También hay otras posibilidades, como:
    - SQL Server
    - Oracle
    - PostgreSQL

# 1. Desarrollo web del lado del servidor

- **DHTML** (Dynamic HTML) no existe como tal, sino que es la conjunción de tres lenguajes:
  - HTML
  - CSS
  - JavaScript
- DHTML sirve para crear las páginas web por las que navegamos todos los días. Cada uno de sus lenguajes componentes se encarga de hacer una parte del trabajo.

# 2. Caja de herramientas

Resumiendo, las herramientas que necesitaremos son:

- HTML
- CSS
- JavaScript / jQuery / Ajax
- PHP u otro lenguaje de script de servidor (Python, Ruby, Perl, etc)
- Un framework para trabajar con el lenguaje elegido.
- MySQL / MariaDB u otro SGBD que permita acceso remoto.

A continuación introduciremos las herramientas básicas. A lo largo del curso, iremos viendo el resto.

# 2. Caja de herramientas: HTML

- **HTML** = HyperText Markup Language (Lenguaje de Etiquetas de Hipertexto)
- Es un **lenguaje para formatear documentos**:
  - Permite definir el tipo de letra, tamaño, formato y color de los textos.
  - Permite insertar imágenes y otro contenido multimedia.
  - Permite crear listas, tablas, enumeraciones...
  - Permite crear enlaces entre secciones del mismo documento, o enlaces con otros documentos (hipertexto)
- **NO** es un **lenguaje de programación**:
  - No permite programar algoritmos.
  - Pero sí permite incrustar otros lenguajes de programación en su interior, aumentando así su potencia.
  - Los trozos de código embebidos dentro de HTML se denominan *scripts.*

# 2. Caja de herramientas: HTML

- En 1990 se crea **HTML** (procedente de un lenguaje anterior, **SGML**) junto con la **World Wide Web**, para formatear los documentos de la www.
- Se amplía en sucesivas versiones hasta la 3.0, que no consiguió éxito debido a las limitaciones de los navegadores de la época.
- Comienza la ***guerra de navegadores***: Microsoft y Netscape sacan sus propios "dialectos" de HTML y destrozan en estándar.
- A partir de **HTML 4** se intenta unir las características de los dos, pero el resultado es demasiado complejo.
  - Se hace evidente que hay que hacer una "limpieza" de HTML
  - Así surge **XHTML**, la versión XML de HTML, mucho más estricta y formal, con menos añadidos pero igual de potente.

# 2. Caja de herramientas: HTML

- Versiones actuales de HTML recomendadas:
  - **HTML 4.01 transicional**: HTML clásico, con todos los elementos del HTML antiguo, aunque se recomiende no usar muchos de ellos)
  - **HTML 4.01 estricto**: también llamado **XHTML**, no permite usar los elementos HTML desaprobados, tales como definición de formatos.
  - **HTML5**: elimina definitivamente los elementos antiguos del lenguaje e incorpora algunos nuevos para completar la asimilación con XML.
  - La especificación para HTML6 (o HTML Next) está actualmente en desarrollo.

# 2. Caja de herramientas: CSS

- **CSS** = Cascade Style Sheet (Hojas de estilo en cascada)
- Es un lenguaje para la **definición de los formatos** utilizados en una página web.
  - CSS sólo permite definir el formato (es decir, el aspecto) de la página, no su contenido.
  - Al definir los formatos en otra parte, se pueden reutilizar a lo largo de una o incluso de varias páginas.
  - Si cambiamos la definición CSS del formato, se cambian automáticamente los formatos de todas las páginas que usen esa definición.
  - El objetivo último es separar completamente el formato de la página de su **contenido** → ***XHTML***.
  - **CSS 2.1** se usa con HTML 4. **CSS3** se usa con HTML5 y ya se puede considerar soportado universalmente.

# 2. Caja de herramientas: JavaScript

- **JavaScript** es un lenguaje interpretado que puede ser incrustado dentro del código HTML de una página web.
- Todos los navegadores web actuales son capaces de interpretar código JavaScript.
- El código JavaScript puede interactuar y modificar cualquier parte del documento HTML, por lo que dota a las páginas web de dinamismo e interactividad.
- *JavaScript no es Java*, aunque su sintaxis está a medio camino entre C++ y Java.
- La implementación de JavaScript de cada navegador es distinta, obteniéndose resultados que no siempre son iguales. Por ejemplo:
  - V8 = motor JS de Chrome
  - WebKit = motor JS de Safari
  - Rhino = motor JS de Mozilla Firefox
  - WebKit = motor JS de Microsoft Edge

# 2. Caja de herramientas: PHP

- PHP es un acrónimo recursivo. Significa “PHP Hypertext Preprocessor”
- Es un **lenguaje de programación** usado generalmente para generar páginas web dinámicas.
  - Es ese caso, aparece embebido en documentos **HTML** / **XHTML**.
  - Pero también puede usarse para crear **aplicaciones convencionales** (usando las extensiones **PHP-Qt** o **PHP-GTK**)
- Permite conectarse con **múltiples bases de datos**: **MySQL**, Oracle, Postgres, SQL Server, DB2, etc. También puede conectar por **ODBC**.
- **Se parece mucho a otros lenguajes** 3GL y O.O. (en particular a C/C++), por lo que la curva de aprendizaje para los que ya saben programar es muy corta.

# 2. Caja de herramientas: PHP

- Surge en 1995 como extensión de CGI (otro lenguaje para acceso a funciones del servidor)
- **PHP3** (1998) tuvo un gran éxito comercial.
- **PHP4** (2000) es la versión más extendida (por desgracia): la mayoría de los scripts en PHP que circulan por la red están escritos en esta versión obsoleta.
- **PHP5** (2004) tiene soporte para orientación a objetos y una biblioteca de clases bastante bien diseñada. Por lo tanto, desde esta versión PHP pasa de ser un lenguaje estructurado (3GL) a ser un lenguaje orientado a objetos.
- **PHP6** empezó a desarrollarse en 2007 y se canceló en 2014.
- **PHP7** es la última versión (7.3.9 en septiembre de 2019). El mantenimiento de PHP4 ha concluido y el de PHP5 se detendrá en 2018, por lo que *todas las nuevas aplicaciones deberían escribirse en PHP7*.

# 2. Caja de herramientas: PHP

- **Lo nuevo en PHP 7:**
  - Mejoras importantes de rendimiento.
  - Unificación de la sintaxis de las variables.
  - Declaración de tipos devueltos por los métodos.
  - Declaración de tipos escalares (integer, float, string y boolean)
  - Clases anónimas.
  - Reemplazo de antiguos errores internos de PHP por excepciones manejables en tiempo de ejecución.
  - Operador de comparación <=>
  - Etc

# 2. Caja de herramientas: PHP

- **Pros**
  - Completamente libre y abierto.
  - Muy eficiente.
  - Ejecutable en (casi) cualquier servidor.
  - Excelente documentación.
  - Curva de aprendizaje baja si ya sabes programar.
  - Entornos de desarrollo abundantes, para todos los gustos.
  - Fácil interoperatibilidad con otros sistemas, en particular con bases de datos.
  - Comunidad muy grande.
  - Sigue siendo líder del mercado de aplicaciones web.

# 2. Caja de herramientas: PHP

- **Cons**
  - Fallos de diseño (corregidos en su mayoría a partir de PHP 5), como:
    - Los métodos para acceso a bases de datos cambian según el SGBD usado.
    - Nombres de funciones inconsistentes.
  - No es completamente orientado a objetos.
  - Tipado confuso y, a veces, impredecible.
  - Grandes (e incompatibles) cambios entre versiones.
  - Pérdida lenta pero imparable de cuota de mercado (en favor de Python)
  - Pésima relación señal/ruido en la web: ¡hay *demasiados* malos desarrolladores en PHP!

# 2. Caja de herramientas: PHP

- **Lenguajes *script* de cliente**
  - Son lenguajes que se ejecutan en la máquina cliente.
  - El servidor web envía al cliente una página HTML con código en otro lenguaje en su interior.
  - El navegador del cliente ejecuta ese código en su máquina
  - **JavaScript es un lenguaje de cliente**
- **Lenguajes *script* de servidor**
  - Son lenguajes que se ejecutan en la máquina servidor.
  - El servidor web ejecuta el script, cuya salida es un fichero en HTML.
  - Ese fichero HTML es enviado a la máquina cliente, que lo interpreta y visualiza. Puede contener en su interior scripts de cliente.
  - **PHP es un lenguaje de servidor**
  - Los scripts de servidor pueden acceder a los recursos ubicados en el servidor: bases de datos, ficheros, etc.

# 2. Caja de herramientas: MariaDB

- **MariaDB** es un gestor de bases de datos relacional multiusuario y multiplataforma.
  - Permite **mútiples conexiones remotas**.
  - El **software libre**.
  - Existen **librerías** para acceder a MariaDB desde muchos lenguajes: C/C++, Java, PHP, Perl, Pascal... Además, hay drivers ODBC.
  - Está muy extendida en aplicaciones web, generalmente en combinación con **PHP**.
  - Cuenta un un interfaz gráfico programado en PHP, llamado **PHPMyAdmin**, que se ejecuta en el navegador web.

# 2. Caja de herramientas: MariaDB

- MySQL surgió como un proyecto OpenSource en Suecia en 1995.
- El objetivo era lograr un SGBD rápido y fiable que cumpliera con el estándar SQL.
- Las primeras versiones (que se denominaron mSQL) eran muy ineficientes.
- La popularización de PHP y su ganancia en eficiencia a partir de la versión 3 la han hecho muy popular en la actualidad.
- Tras su adquisición por Oracle, se intentó relegar al segmento medio-bajo en el mercado de los SGBD y surgió un *fork*: **MariaDB**.
- Versión más reciente (mayo 2019): MariaDB 10.4.5

# 3. Sintaxis de PHP

## 3. Sintaxis de PHP

- El código PHP se escribe **incrustado** dentro de un documento de texto mediante estas etiquetas:

<?php .... ?>

  - La sintaxis clásica se ha desechado en PHP 7:

<script language= “php”> ... </script>

- Este archivo debe tener **extensión .php**.
- El servidor ejecuta el código PHP que encuentre dentro del archivo, mientras que el código HTML es enviado al cliente sin modificar.

# 3. Sintaxis de PHP

- **Comentarios** en PHP:

```
// Comentario de una línea
#  Comentario de una línea
/* Comentario de una o varias líneas */
```

- **Operadores**: son iguales que los de C/C++:
  - Asignación:  $a = 3;
  - Comparación:  ==, <=, >=, !=, <=>, etc.
  - Operadores aritméticos: +, -, *, /, %...
  - Operadores lógicos: &&, ||, !
  - Etc.

# 3. Sintaxis de PHP

- **Variables**
  - El identificador siempre debe empezar por $
  - No es necesario declararlas: al inicializarlas queda especificado el tipo. En PHP 7 pueden indicarse los tipos predefinidos (int, float, string...)
  - Ejemplos:

```
$a = 4;                          // Variable entera (PHP 5)
int $a = 4;                      // Variable entera (PHP 5 o 7)
$media = 52.75;                  // Variable real
$texto = “Hoy es lunes”;         // Variable string
```

- **Variables asignadas por referencia (&)**:
  - Cuando una variable se asigna a otra usando el operador &, ambas pasan a compartir el mismo espacio de memoria. A partir de ahora, un cambio en una de las dos provoca un cambio en la otra.

```
$a = &$b;          // a y b son “la misma” variable
```

# 3. Sintaxis de PHP

- **Cambio de tipo en las variables**
  - Cualquier variable puede cambiarse de tipo con la función **setType**:

```
$a = “10”;                  // a es una cadena
setType($a, “integer”);     // a se convierte a entero
```

  - Los **tipos predefinidos** en PHP son:

integer (entero)
double (real)
bool (booleano)
string (cadena)
array (pues eso)

# 3. Sintaxis de PHP

- **Arrays**
  - Los arrays en PHP son colecciones de variables del mismo o de distinto tipo identificadas por un índice.
  - Ejemplos:

```
$a[1] = “lunes”;
$a[2] = “martes”;
$a[3] = “miércoles”;
```

  - Lo habitual es que el índice sea un número entero, pero puede no serlo (array asociativo):

```
$a[“ESP”] = “España”;
$a[“FRA”] = “Francia”;
$a[“POR”] = “Portugal”;
```

# 3. Sintaxis de PHP

- **Condicionales**

```
if (condición)
{
    acciones-1;
}
else
{
    acciones-2;
}
```

# 3. Sintaxis de PHP

- **Bucle mientras**

```
while (condición)
{
    acciones;
}
```

- **Bucle repetir**

```
do
{
    acciones;
}
while (condición);
```

# 3. Sintaxis de PHP

- **Bucle para**
  - El bucle para controlado por contador es idéntico a C/C++

```
for (inicialización; condición; incremento)
{
    acciones;
}
```

  - Hay una variedad muy interesante: el bucle **foreach** para recorrido de arrays asociativos:

```
foreach ($array as $índice=>$variable)
{
    acciones;
}
```

El bucle *foreach* se repite una vez para cada valor guardado en el array. Ese valor se asigna a la variable en cada repetición.

# 3. Sintaxis de PHP

- **Funciones y procedimientos (¡sólo en PHP4!)**
  - Los subprogramas (funciones y procedimientos) se escriben en PHP con la misma palabra: **function**.
  - Las funciones deben devolver un valor en su última línea con **return**. Los procedimientos no.
  - Los parámetros de la función en PHP siembre se pasan **por valor**. Si un procedimiento tiene que devolver varios valores, lo hará en un return con un array, como veremos en los ejercicios.
  - **Ejemplo**:

```
function calcular_iva($base, $porcentaje)
{
        $total = $base * $porcentaje /100;
        return $total;
}
```

# 3. Sintaxis de PHP

- **Clases y objetos (¡solo en PHP5 y PHP7!)**

```
class miClase
{
    // Declaración de propiedades (atributos)
    public $var = 'soy una variable de clase';

    // Declaración de métodos
    public function mostrarVar() {
        echo $this->var;
    }
    private function resetVar() {
       $this->var = '';
    }
}
// ----------------------------------
$miObjeto = new miClase();
$miObjeto->mostrarVar();
```

# 3 Sintaxis de PHP

- **Ámbito de las variables: paso de parámetros por la URL (1)**
  - Las variables de una función/clase/método PHP son **locales**, es decir, sólo están disponibles en esa función/clase/método.
  - Si se definen variables *fuera* de una función, serán **globales** a todo el fichero actual, pero no pueden usarse en scripts ubicados en otros ficheros.
  - Para **compartir variables entre scripts diferentes**, se usa habitualmente la URL (o dirección):

```
<a href="pagina.php?
   variable1=valor1&variable2=valor2&etc...">
```

  - El código PHP ubicado en "página.php" puede recuperar el valor de las variables **$variable1**, **$variable2**, etc.

# 3. Sintaxis de PHP

- **Ámbito de las variables: paso de parámetros por la URL (2)**
  - Otra forma de usar las variables recibidas a través de la URL es con el array del sistema **$HTTP_GET_VARS**, que se indexa con el nombre de las variables:
  - Ejemplo:

```
<?php
    echo “La variable 2 vale:
      $HTTP_GET_VARS[“variable2”]<br>”
?>
```

  - **Nota**: a partir de la versión PHP 4.1 existe la abreviatura **$_GET**.

# 3. Sintaxis de PHP

## Salida de datos

- Recuerda que PHP se ejecuta dentro de un navegador web. Por lo tanto, su salida debe poder verse en el navegador.
- El navegador web sólo puede mostrar páginas escritas en código HTML, por lo que ***PHP debe producir como salida código HTML o XHTML válido***.
- **Ejemplo** (observa el uso de "**echo**" para producir la salida):

```
<body>
    <?php
        echo "Soy un script de PHP y estoy generando
                código HTML. Para demostrarlo
                voy a escribir <b>esto en negrita</b>"
    ?>
</body>
```

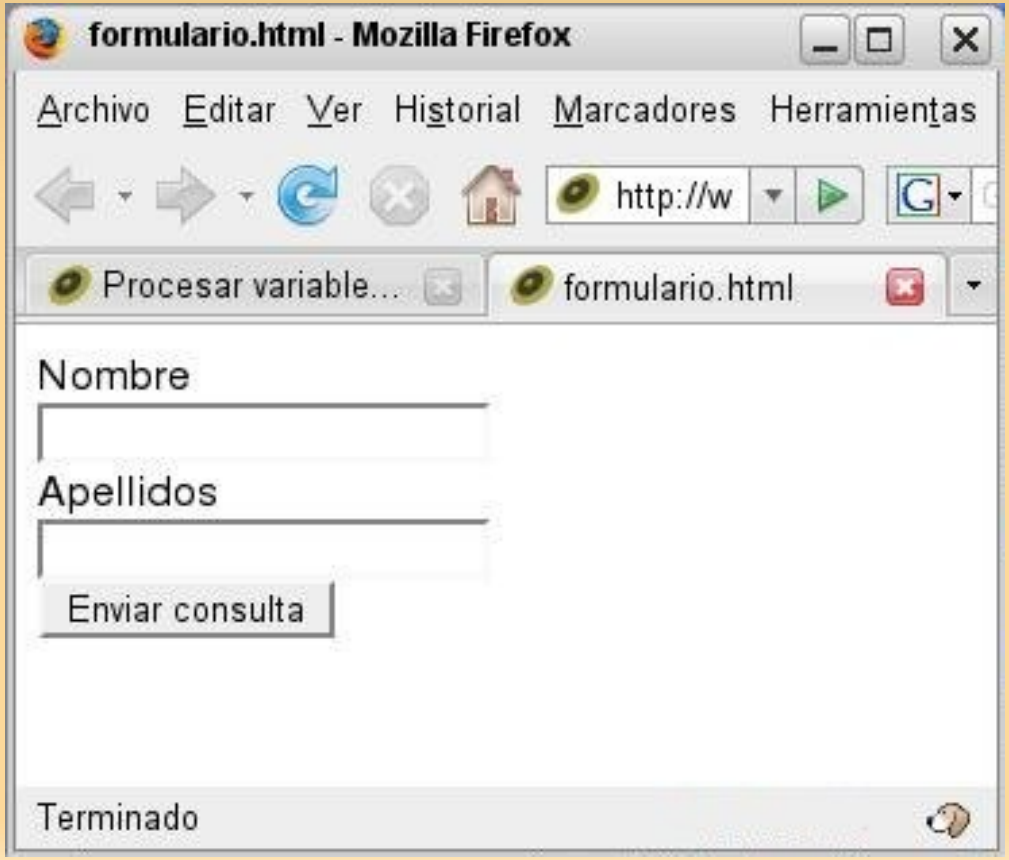
# 3. Sintaxis de PHP

- **Entrada de datos a través de formulario (1)**
  - Como PHP se ejecuta dentro de HTML, sólo puede recibir datos a través del navegador web.
  - Y sólo hay una forma de introducir datos en una página web: a través de un **formulario**.
  - Ejemplo: supongamos que hemos definido en HTML este sencillo formulario:

```
<body>
    <form method="post" action="destino.php">
        Nombre<br/>
        <input type="text" name="nombre"><br/>
        Apellidos<br>
        <input type="text" name="apellidos"><br/>
        <input type="submit">
    </form>
</body>
```

# 3. Sintaxis de PHP

- **Entrada de datos a través de formulario (2)**
  - El formulario se verá en el navegador más o menos así:



# 3. Sintaxis de PHP

- **Entrada de datos a través de formulario (3)**
  - Al pulsar sobre “Enviar”, se ejecutará el script **destino.php**.
  - Ese script recibirá **dos variables** llamadas **$nombre** y **$apellido**, con el valor que el usuario haya introducido en el formulario, y las podrá usar como cualquier otra variable. P. ej:

```
<?php echo “La variable nombre vale $nombre<br>” ?>
```

  - También se puede acceder a las variables con el array del sistema **$HTTP_POST_VARS**, indexándolo con el nombre de la variable:

```
<?php echo “La variable nombre vale
  $HTTP_POST_VARS[“nombre”]<br>” ?>
```

  - Desde PHP 4.1, ese array puede abreviarse como **$_POST**.

# 4. Interacción de PHP con MySQL

## 4. Interacción de PHP con MySQL o MariaDB

- **MySQL** es un **SGBD profesional**, por lo que la interacción con él busca ser eficiente y segura, pero no necesariamente fácil.
- Hay básicamente tres métodos de utilizar MySQL:
  - **A través de la línea de comandos**:
    - Iniciamos una sesión en MySQL con:

      $ mysql -u *nombre_usuario* -p *contraseña*
    - Y luego tenemos a nuestra disposición montones de comandos para hacer cosas con la base de datos, incluyendo cualquier instrucción válida en SQL.
  - **A través de un interfaz gráfico como PHPMyAdmin**:
    - Es un conjunto de scripts en lenguaje PHP que proporcionan un interfaz aceptablemente cómodo para trabajar con MySQL.
    - Es el método más utilizado para ejecución interactiva de SQL.
  - **A través de un programa** escrito en PHP o algún otro lenguaje con posibilidad de acceso a MySQL.
    - Este método de acceso será el que nosotros practicaremos a continuación.

# 4. Interacción de PHP con MySQL

## Acceso a MySQL con PHP4

- El modo en que se accedía a bases de datos en PHP4 era mediante **bibliotecas de funciones** diferentes para cada SGBD.
- Este tipo de codificación está obsoleta y se desaconseja su uso. Ya no tiene soporte oficial, por lo que no se resolverán futuros problemas de seguridad o estabilidad.
- Lo mostramos aquí para que sepáis lo que **NO se debe hacer.**
- Encontraréis mucho código de esta naturaleza en la red que **DEBE SER EVITADO.**

# 4. Interacción de PHP con MySQL

## Acceso a MySQL con PHP4

- PHP4 utiliza una biblioteca de **funciones** PHP cuyo nombre empieza por mysql_.
- Por ejemplo, para insertar un registro en una BD MySQL:

```
<?php
    //Conectamos con MySQL
    mysql_connect("URL","nombre_usuario","contraseña");

    //Selección de la base de datos con la que vamos a trabajar
    mysql_select_db("nombre_base_de_datos");

    //Ejecucion de cualquier sentencia SQL válida
    mysql_query("INSERT INTO clientes (nombre,telefono)
      VALUES ('$nombre','$telefono')");
?>
```

# 4. Interacción de PHP con MySQL

## Consultas SQL con PHP4 (1/2)

- La ejecución de consultas (SELECT) produce la devolución de un conjunto de registros. Esos regisotrs se manejan en PHP con un **cursor**. Observa cómo se hace en este ejemplo:

```
<BODY>
    <?php
        //Nos conectamos con MySQL
        mysql_connect(“URL","user","password");

        //Seleccionamos la base de datos con la que vamos a trabajar
        mysql_select_db("nombre_base_datos");

        //Ejecutamos la consulta SQL
        $result=mysql_query("SELECT * FROM Clientes");
    ?>
```

*... continúa en la pág. siguiente ...*

# 4. Interacción de PHP con MySQL

## Consultas SQL con PHP4 (2/2)

*... viene de la pág. anterior ...*

```
<table align="center"><tr>
<th>Nombre</th>
<th>Teléfono</th></tr>
<?
      //Mostramos los registros
      while ($registro=mysql_fetch_array($result)) {
           echo '<tr><td>'.$registro["nombre"].'</td>';
           echo '<td>'.$registro["telefono"].'</td></tr>';
      }
      mysql_free_result($result)
?>
</table>
</BODY>
```

# 4. Interacción de PHP con MySQL

## Otras funciones importantes para trabajar con MySQL en PHP4 (1/2)

- **mysql_connect**(“URL”, “user”, “password”);
  - Conecta con MySQL. En la URL hay que poner la dirección completa dónde se encuentra el servidor funcionando (ejemplos: “localhost”, “219.39.21.23”, “http://miservidor.com”, “/home/user”, etc.)
- **mysql_select_db**(“nombre_base_de_datos”);
  - Abre una base de datos de las que MySQL tenga disponibles.
- **mysql_close**($variable_bd);
  - Cierra la conexión con MySQL. La $variable_bd nos la devolvió mysql_connect().
  - Es aconsejable hacerlo antes de que termine el programa.

# 4. Interacción de PHP con MySQL

## Otras funciones importantes para trabajar con MySQL en PHP4 (2/2)

- **mysql_query("sentencia-SQL");**
  - Ejecuta el código SQL especificado. Pueden introducirse en su interior variables PHP (siempre precedidas del símbolo $).
  - Si es un SELECT, el resultado debe asignarse a una variable.
- **mysql_fetch_array**($variable);
  - Procesa la variable donde se guardó el resultado de un SELECT.
  - Cada llamada devuelve un registro completo, que debe asignarse a su vez a otra variable.
  - Con esa variable podemos acceder a cada uno de los campos, indexándola por el nombre del campo (ver ejemplo anterior).
- **mysql_affected_rows**();
  - Devuelve el número de registros afectados por la última operación SQL (válido para INSERTE, UPDATE y DELETE)
- **mysql_num_rows**();
  - Como la anterior, pero para SELECT

# 4. Interacción de PHP con MySQL

## Acceso a MySQL con PHP5 y PHP7

- Desde PHP5 se utiliza una **biblioteca de clases** para acceder a los diferentes SGBDs.
- Este tipo de codificación es la que se recomienda en la actualidad.
  - Todos los nuevos desarrollos deberían usar las bibliotecas de clases y prescindir de las viejas librerías de funciones.
  - Todos los desarrollos antiguos deberían migrarse a PHP7 por razones de seguridad, compatibilidad y eficiencia.
- **¡Cuidado!** Mucho código de ejemplo de PHP que circula por la red es PHP4 y DEBE SER EVITADO.

# 4. Interacción de PHP con MySQL

## Acceso a MySQL con PHP5 / PHP7

- Por ejemplo, para insertar un registro en una BD MySQL:

```
<?php
    // Conectamos con el servidor y abrimos la BD.
    $conexdb = new
      mysqli('host','username','password','database');

    // Aquí se ejecutaría cualquier sentencia SQL válida.
    // Por ejemplo:
    $conexdb->query("INSERT INTO clientes (nombre,telefono)
      VALUES ('$nombre','$telefono')");
?>
```

# 4. Interacción de PHP con MySQL

## Aviso para navegantes

PHP5/7 proporciona varios mecanismos para acceder a bases de datos (¡demasiadas formas de hacer lo mismo!):

- **La extensión mysqli en su forma procedimental.**
  - Es idéntica a la de PHP4, pero cambiando la palabra “mysql” por “mysqli”.
  - Por ejemplo, mysql_connect() cambia a mysqli_connect().
  - Apta para programadores perezosos y anticuados, que no quieren pasarse a la POO.
- **La extensión mysqli en su forma orientada a objetos**.
  - Es la que nosotros recomendamos. Los ejemplos que estamos mostrando usan esta forma.
- **La extensión PDO**. Se trata de una clase genérica que permite acceder a cualquier gestor de bases de datos mediante el mismo conjunto de métodos. En funcionalidad y rendimiento es idéntica a mysqli.

# 4. Interacción de PHP con MySQL

## Consultas SQL con PHP5 y PHP7 (1/2)

- La ejecución de consultas (SELECT) produce la devolución de un conjunto de registros. Esos registros se manejan en PHP con un **cursor**. Observa cómo se hace en este ejemplo:

```
<BODY>
    <?php
        //Nos conectamos con MySQL
        $db = new mysqli(“URL","user","password", “database”);

        // Comprobamos que la conexión se ha realizado
        if($db->connect_error){
                die(“Error en la conexion : “.$conexion->connect_error);
            }

        //Ejecutamos la consulta SQL
        $result=$db->query("SELECT * FROM Clientes");
    ?>
```

*... continúa en la pág. siguiente ...*

# 4. Interacción de PHP con MySQL

## Consultas SQL con PHP5 y PHP7 (2/2)

*... viene de la pág. anterior ...*

```
<table align="center"><tr>
<th>Nombre</th>
<th>Teléfono</th></tr>
<?
      //Mostramos los registros
      while ($registro=$result->fetch_array()) {
            echo '<tr><td>'.$registro["nombre"].'</td>';
            echo '<td>'.$registro["telefono"].'</td></tr>';
      }
      $db->free($result);  // Libera la memoria usada por el cursor
      $db->close();   // Cierra la conexión con el servidor
?>
</table>
</BODY>
```