Parte I : El Lenguaje PHP

CAPITULO I: INTRODUCCIÓN

1.1. ¿Qué es PHP?

PHP (acronimo de "PHP: Hypertext Preprocessor" - Preprocesador de Hipertexto) es un lenguaje "open source" interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor web.

Con estas nos referimos a un lenguaje de programación que está muy orientado al desarrollo de aplicaciones web. Cuando pedimos a nuestro servidor de web una página PHP, que no es más que un programa PHP que genera HTML, antes de enviar dicha página al cliente se la pasa al interprete de PHP. Este la interpreta y es el resultado de esta interpretación del programa PHP, contenido en la página PHP, lo que termina llegando al cliente.

Supongamos que el contenido de una página web que reside en el servidor, y cuyo nombre es "intro.php", tiene el siguiente contenido:

```
<? echo "<h1>Mensaje desde PHP</h1>"; ?>
```

Cuando un navegador le pida al servidor de web la página "intro.php", el servidor de web va a darse cuenta, por la extensión ".php", de que esta página ha de enviarse primero al intérprete de PHP. Este recibe el contenido de la página y como resultado de esta ejecución (interpretación) genera una página HTML, que es la que envía al cliente a través de Apache u otros servidor web.

Como veremos en una página PHP se puede mezclar HTML y PHP, algo muy flexible pero que hay que manejar con cuidado ya que puede llevar a confusiones y, sobretodo, a que el equipo que diseñe las páginas web y el que programe la aplicación no puedan ser independientes.

1.2. Diferencias entre ASP y PHP

Active Server Pages	**PHP: Hypertext Preprocessor**
• Software propietario • Plataformas Microsoft • Varios lenguajes (VBScript, JavaScript)	• Free Software • Multiplataforma • Un solo lenguaje: PHP

1.3. Características de PHP

- Más rápido que ASP
- Lenguaje más fácil y potente
- Integración perfecta con 8 servidores HTTP
- Acceso a 20 tipos de Bases de Datos
- Diseño modular de fácil ampliación
- Licencia abierta

1.4. Historia y Desarrolladores

Fechas

- Inicio del desarrollo en otoño de 1994
- PHP Versión 1 en primavera 1995
- PHP Versión 2 1995-1997
- PHP Versión 3 1997-2000
- PHP Versión 4 en el segundo trimestre de 2000
- PHP Versión 4.2.x 2002 y próximo PHP 5.0

Equipo de Desarrollo (195 personas con acceso al CVS)

- Zeev Suraski y Andi Gutmans (Israel)
- Shane Caraveo (Florida)
- Stig Bakken (Norway)
- Andrei Zmievski (Lincoln, Nebraska)
- Sascha Schumann (Dortmund, Germany)
- Thies C. Arntzen (Hamburg, Germany)
- Jim Winstead (Los Angeles)
- Sam Ruby (Raleigh, NC)
- Rasmus Lerdorf (San Francisco)

1.5. Bases de datos soportadas

SQL

- IBM DB2
- Informix
- MySQL
- ODBC
- Oracle
- PostgreSQL
- Sybase, Interbase/Firebird, Otros.

1.6. Extensión de los ficheros

- .php3 Indica código PHP versión 3.x.
- .php4 Indica código PHP versión 4.x.
- .php Indica código PHP. Esta extensión es más genérica.
- .phtml Actualmente en desuso.

1.7. Un Primer Script en Php

```
<html>
<body>
<?php
        echo  "Hola. Este es mi primer script en PHP \n";
?>
</body>
</html>
```

Una vez escrito esto lo grabamos en un fichero con la extensión **php (**ejemplo **test.php)**, y lo colocamos en nuestro servidor web (directorio).

Para visualizarlo cargamos un navegador y escribimos http://mi_servidor/test.php, se visualizará el texto

Hola. Este es mi primer script en PHP.

Lo primero que apreciamos en el script son sus delimitadores. En la primera línea del script vemos **<?php** que nos indica que comienza un script en PHP, y en la última colocamos **?>** para indicar el final del script. Hay que destacar que todas las líneas que se encuentre entre estos delimitadores deben acabar en **punto y coma**, excepto las sentencias de control (if, switch, while, etc.).

Como en toda programación, es importante poner muchos comentarios, para lo cual si queremos comentar una sola línea tenemos que poner al principio de la línea **//**, si lo que queremos es comentar varias utilizaremos los delimitadores **/* - */**.

Para que el servidor envié texto utilizaremos la instrucción **echo**, aunque también podemos utilizar **printf** de uso similar al del **C** o **Perl**.

Existen múltiples formas de incluir código PHP:

a. <?php echo("Hola a todos"); ?>

b. <? echo("Hola a todos "); ?>

c.
```
<script language="php">
    echo("Hola a todos ");
</script>
```

d. <% echo("Hola a todos "); %>

Puede ser, que no todas estén disponibles en su sistema, esto depende de la instalación que haya realizado. Ante la duda, utilice la primera forma.

Comentarios

Los comentarios en PHP se pueden poner en varios formatos, de tipo C, C++ y Shell. Si bien se puede hacer, no es recomendable mezclar distintos tipos de comentario en un archivo, sino elegir una sintaxis y quedarse con ella durante todo el documento.

```
<?
echo( "Hola Fabian" );

/* comentario de
 varias líneas */


# comentario de una línea


// comentario de una línea
?>
```

Como en la mayoría de los lenguajes, no se pueden poner comentarios dentro de otros comentarios.

CAPITULO II: Variables y Operadores

2.1. Variables

Todas las variables deben ir precedidas por el **signo dólar** ($), y le asignamos contenido con el **signo igual** (=). PHP **distingue** entre mayúsculas y minúsculas, por lo que no es lo mismo $myvar que $Myvar, éstas son dos variables totalmente distintas.

```
<html>
<body>
<?php

  $myvar = "CHIMBOTE \n";
 $Myvar = "SANTA \n";

  //Esto imprimirá CHIMBOTE
  echo $myvar;

  //Esto imprimirá SANTA
  ECHO $Myvar;
?>
</body>
</html>
```

Se observa las dos formas de escribir **echo**, en mayúsculas y en minúsculas, **PHP no las distingue** a la hora de usar funciones o sentencias del lenguaje.

El uso de la barra invertida, como en \n, no es obligatorio, pero ayuda a la depuración del código que enviamos al navegador, además del \n existen otros usos:

\" Carácter dobles comillas
**** Carácter barra invertida
\n Nueva línea
\r Retorno de carro
\t Tabulador horizontal

Normalmente PHP elegirá un tipo apropiado de acuerdo al contexto (o dato/expresión asignado) para cada variable:

```
$a = 123;    # entero
$a = 123.1;  # flotante
$a = "abc";  # string
```

2.2. Constantes

Las constantes son similares a las variables, con la salvedad de que no llevan el **signo dólar** delante, y sólo la podemos asignar una vez. Para definir una constante usaremos la función **define** como sigue:

```
<html>
<body>

        <?php

        define ("CONSTANTE", "Hola Mundo");
        printf (CONSTANTE);

        ?>

</body>
</html>
```

PHP crea diversas constantes al arrancar, como PHP_VERSION que contiene la versión de PHP, TRUE que le asigna 1 o FALSE que le asigna 0.

2.3. Operadores Aritméticos

$a + $b	Suma
$a - $b	Resta
$a * $b	Multiplicación
$a / $b	División
$a % $b	Resto de la división de $a por $b
$a++	Incrementa en **1** a $a
$a--	Resta **1** a $a

2.4. Operadores de Cadenas

El único operador de cadenas que existen es el de concatenación, el **punto**.

```
$a = "Hola";
$b = $a . "Mundo";  // Ahora $b contiene "Hola Mundo"
```

En este punto hay que hacer una distinción, la interpretación que hace PHP de las **simples y dobles comillas**. En el segundo caso PHP interpretará el contenido de la cadena.

```
$a = "Mundo";
echo = 'Hola $a'; //Esto escribirá "Hola $a"
echo = "Hola $a"; //Esto escribirá "Hola Mundo"
```

2.5. Operadores de Comparación

$a < $b	$a **menor que** $b
$a > $b	$a **mayor que** $b
$a <= $b	$a **menor o igual que** $b
$a >= $b	$a **mayor o igual que** $b
$a == $b	$a **igual que** $b
$a != $b	$a **distinto que** $b

2.6. Operadores Lógicos

AND

$a **&&** $b Verdadero si ambos son verdadero

OR

$a **||** $b Verdadero si alguno de los dos es verdadero

!$a Verdadero si $a es falso, y recíprocamente

2.7. Operadores de Asignación

$a = $b	**Asigna** a $a el contenido de $b
$a += $b	Le **suma** a $b a $a
$a -= $b	Le **resta** a $b a $a
$a *= $b	**Multiplica** $a por $b y lo asigna a $a
$a /= $b	**Divide** $a por $b y lo asigna a $a
$a .= $b	**Añade** la cadena $b a la cadena $a

CAPITULO III: Estructuras de Control

Las sentencias de control permiten ejecutar bloque de códigos dependiendo de unas condiciones. Para PHP el 0 es equivalente a Falso y cualquier otro número es Verdadero.

3.1. IF...ELSE

La sentencia **IF...ELSE** permite ejecutar un bloque de instrucciones si la condición es **Verdadera** y otro bloque de instrucciones si ésta es **Falsa.** Es importante tener en cuenta que la condición que evaluemos ha de estar encerrada entre **paréntesis** (esto es aplicable a todas las sentencias de control).

```
if (condición) {
        Este bloque se ejecuta si la condición es VERDADERA
} else {
        Este boque se ejecuta si la condición es FALSA
}
```

Existe una forma sencilla de usar la sentencia IF cuando no tenemos que usar el ELSE y solo tenemos que ejecutar una línea de código.

```
if ($a > 4) echo "$a es mayor que 4";
```

3.2. IF...ELSEIF...ELSE

La sentencia IF...ELSEIF...ELSE permite ejecutar varias condiciones en cascada. Para este caso veremos un ejemplo, en el que utilizaremos los operadores lógicos.

```
if ($nombre == ""){ echo "Tú no tienes nombre";
} elseif (($nombre=="eva") OR ($nombre=="Eva")) {
   echo "Tu nombre es EVA";
} else {
echo "Tu nombre es " . $nombre;
}
```

3.3. SWITCH...CASE...DEFAULT

Una alternativa a IF...ELSEIF...ELSE, es la sentencia SWITCH, la cuál evalúa y compara cada expresión de la sentencia CASE con la expresión que evaluamos, si llegamos al final de la lista de CASE y encuentra una condición Verdadera, ejecuta el código de bloque que haya en DEFAULT. Si encontramos una condición verdadera debemos ejecutar un BREAK para que la sentencia SWITCH no siga buscando en la lista de CASE. Veamos un ejemplo.

```
<?php
switch ($dia) {
case "Lunes":
        echo "Hoy es Lunes";
        break;
case "Martes":
        echo "Hoy es Martes";
        break;
// codifique los demás días …
case "Sábado":
        echo "Hoy es Sábado";
        break;
case "Domingo":
        echo "Hoy es Domingo";
        break;
default:
        echo "Esa cadena no corresponde a ningún día de la semana";
}
?>
```

3.4. WHILE

La sentencia WHILE ejecuta un bloque de código mientras se cumpla una determinada condición.

```
<?php
$num = 1;
while ($num < 5) {
        echo $num;
        $num++;
}
?>
```

Podemos romper un bucle WHILE utilizando la sentencia **BREAK**.

```
<?php
$num = 1;
while ($num < 5) {
        echo $num;
        if ($num == 3){
        echo "Aquí nos salimos \n";
        break;
        }
        $num++;
}
?>
```

3.5. DO...WHILE

Esta sentencia es similar a WHILE, salvo que con esta sentencia primero ejecutamos el bloque de código y después se evalúa la condición, por lo que el bloque de código se ejecuta siempre al menos una vez.

```
<?php
$num = 1;
do {
        echo $num;
        if ($num == 3){
                echo "Aquí nos salimos \n";
                break;
        }
        $num++;
} while ($num < 5);
?>
```

3.6. FOR

El bucle FOR no es estrictamente necesario, cualquier bucle FOR puede ser sustituido fácilmente por otro WHILE. Sin embargo, el bucle FOR resulta muy útil cuando debemos ejecutar un bloque de código a condición de que una variable se encuentre entre un valor mínimo y otro máximo. El bucle FOR también se puede romper mediante la sentencia **BREAK**.

```
<?php
for ($num = 1; $num <=5; $num++){
        echo $num;
        if ($num == 3){
                echo "Aquí nos salimos \n";
                break;
        }
}
?>
```

CAPITULO IV: Tablas - Arrays

4.1. Tablas Unidimensionales

Las tablas (o array en inglés), son muy importantes en PHP, ya que generalmente, las funciones que devuelven varios valores, como las funciones ligadas a las bases de datos, lo hacen en forma de tabla. En PHP disponemos de dos tipos de tablas.

El primero sería el clásico, utilizando índices:

```
<?php
        $ciudad[] = "París";
        $ciudad[] = "Roma";
        $ciudad[] = "Sevilla";
        $ciudad[] = "Londres";
        printf ("yo vivo en " . $ciudad[2] . "<BR>\n");
?>
```

Esta es una forma de asignar elementos a una tabla, pero una forma más formal es utilizando la función array:

```
<?php
        $ciudad = array("París", "Roma", "Sevilla", "Londres");

        //contamos el número de elementos de la tabla
        $numelentos = count($ciudad);

        //imprimimos todos los elementos de la tabla
        for ($i=0; $i < $numelentos; $i++)
        {
                printf ("La ciudad $i es $ciudad[$i] <BR>\n");
        }
?>
```

Sino se especifica, el primer índice es el **cero**, pero podemos utilizar el operador => para especificar el índice inicial.

```
$ciudad = array(1=>"París", "Roma", "Sevilla", "Londres");
```

Un segundo tipo, son las **tablas asociativas**, en las cuáles a cada elemento se le asigna un valor (key) para acceder a él.

Para entenderlo, que mejor que un ejemplo, supongamos que tenemos una tabla en la que cada elemento almacena el número de visitas a nuestra web por cada día de la semana.

Utilizando el método clásico de índices, cada día de la semana se representaría por un entero, 0 para lunes, 1 para martes, etc.

```
$visitas[0] = 200;
$visitas[1] = 186;
```

si usamos las tablas asociativas sería

```
$visitas["lunes"] = 200;
$visitas["martes"] = 186;
```

o bien,

```
$visitas = array("lunes"=>200; "martes"=>186);
```

Ahora bien, recorrer una tabla y mostrar su contenido es sencillo utilizando los índices, pero ¿cómo hacerlo en las tablas asociativas?. La manipulación de las tabas asociativas se hace a través de funciones que actúan sobre un puntero interno que indica la posición. Por defecto, el puntero se sitúa en el primer elemento añadido en la tabla, hasta que es movido por una función:

current - devuelve el valor del elemento que indica el puntero
pos - realiza la misma función que **current**
reset - mueve el puntero al **primer** elemento de la tabla
end - mueve el puntero al **último** elemento de la tabla
next - mueve el puntero al elemento **siquiente**
prev - mueve el puntero al elemento **anterior**
count - devuelve el número de elementos de una tabla.

Veamos un ejemplo de las funciones anteriores:

```
<?php
	$semana = array("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo");
	echo count($semana); //7

	//situamos el puntero en el primer elemento
	reset($semana);
	echo current($semana); //lunes
	next($semana);

	echo pos($semana); //martes
	end($semana)

	echo pos($semana); //domingo
	prev($semana);

	echo current($semana); //sábado
?>
```

Recorrer una tabla con las funciones anteriores se hace un poco lioso, para ello se recomienda utilizar la función each().

```
<?php
$visitas = array("lunes"=>200, "martes"=>186, "miércoles"=>190, "jueves"=>175);
reset($visitas);

while (list($clave, $valor) = each($visitas))
{
	echo "el día $clave ha tenido $valor visitas<BR>";
}
?>
```

La función each() devuelve el valor del elemento actual, en este caso, el valor del elemento actual y su clave, y desplaza el puntero al siguiente, cuando llega al final devuelve **FALSO**, y termina el bucle while().

4.2. Tablas multidimensionales

Las tablas multidimensionales son simplemente tablas en las cuales cada elemento es a su vez otra tabla.

```
<?php
$calendario[] = array (1, "enero", 31);
$calendario[] = array (2, "febrero", 28);
$calendario[] = array (3, "marzo", 31);
$calendario[] = array (4, "abril", 30);
$calendario[] = array (5, "mayo", 31);
while (list($clave, $valor ) = each($calendario)){
{
	$cadena = $varlor[1];
	$cadena .= " es el mes número " . $valor[0];
	$cadena .= "y tiene " . $varlor[2] . " días<BR>";
	echo $cadena;
}
?>
```

La función list() es más bien un operador de asignación, lo que hace es asignar valores a unas lista de variables. En este caso los valores son extraídos de una tabla por la función each().

CAPITULO V: Funciones

5.1. Las Funciones

Muchas veces, cuando trabajamos en el desarrollo de una aplicación, nos surge la necesidad de ejecutar un mismo bloque de código en diferentes partes de nuestra aplicación. Una Función no es más que un bloque de código al que le pasamos una serie de parámetros y nos devuelve un valor. Como todos los lenguajes de programación, PHP trae una gran cantidad de funciones para nuestro uso, pero las funciones más importantes son las que nosotros creamos.

Para declarar una función debemos utilizar la instrucción **function** seguido del nombre que le vamos a dar, y después entre paréntesis la lista de argumentos separados por comas, aunque también habrá funciones que no recogan ningún argumento.

```
function nombre_de_funcion (arg_1, arg_2, ..., arg_n)

{

    bloque de código

}
```

Cualquier instrucción válida de PHP puede aparecer en el cuerpo (lo que antes hemos llamado bloque de código) de una función, incluso otras funciones y definiciones de clases.

En PHP no podemos redefinir una función previamente declarada, y además en PHP3, las funciones deben definirse siempre antes de que se invoquen, en PHP4 este requerimiento ya no existe.

5.2. La instrucción RETURN

Cuando invocamos una función, la ejecución del programa pasa a ejecutar las líneas de código que contenga la función, y una vez terminado, el programa continua su ejecución desde el punto en que fue llamada la función.

Existe una manera de terminar la ejecución de la función aunque aún haya código por ejecutar, mediante el uso de la instrución return terminamos la ejecución del código de una función y devolvemos un valor. Podemos tener varios return en nuestra función, pero por lo general, cuantos más return tengamos menos reutilizable será nuestra función.

```
<?php

  function mayor ($x, $y)
  {
        if ($x > $y) {
                return $x." es mayor que".$y;
        } else {
                return $y." es mayor que".$x;
        }
  }

?>
```

Aunque quedaría mejor:

Con la instrucción return puede devolverse cualquier tipo de valor, incluyendo tablas y objetos. PHP solo permite a las funciones devolver un valor, y para solventar este pequeño problema, si queremos que nuestra función devuelva varios tenemos que utilizar una tabla (array).

5.3. Parámetros de las funciones

Existen dos formas de pasar los parámetros a una función, por valor o por referencia. Cuando pasamos una variable por ***valor*** a una función, ocurra lo que ocurra en ésta en nada modificará el contenido de la variable. Mientras que si lo hacemos por ***referencia***, cualquier cambio acontecido en la función sobre la variable lo hará para siempre.

En PHP, por defecto, las variables se pasan por valor. Para hacerlo por referencia debemos anteponer un ampersand (&) a la variable.

```
<?php

    function suma ($x, $y)
    {
      $x = $x + 1;
      return $x+$y;
    }

    $a = 1;
    $b = 2;

    //parámetros por valor
    echo suma ($a, $b);    // imprimirá 4
    echo $a;    // imprimirá 1

    //parámetros por referencia
    echo suma (&$a, $b);    // imprimirá 4
    echo $a;      //imprimirá 2

 ?>
```

Si queremos que un parámetro de una función se pase siempre por referencia debemos anteponer un ampersand (&) al nombre del parámetro en la definición de la función.

En PHP podemos definir valores por defecto para los parámetros de una función. Estos valores tienen que ser una expresión constante, y no una variable o miembro de una clase. Además cuando usamos parámetros por defectos, éstos deben estar a la derecha de cualquier parámetro sin valor por defecto, de otra forma PHP nos devolverá un error.

```
<?php

  function suma ($x=1, $y)
  {
    $x = $x + 1;
    return $x+$y;
  }
?>
```

Si ejecutamos esta función nos daría error, ya que hemos dado a $x el valor 1 por defecto y la hemos colocado a la izquierda de un parámetro que no tiene valor por defecto. La forma correcta es:

```
<?php

  function suma ($y, $x=1)
  {
    $x = $x + 1;
    return $x+$y;
  }
?>
```

5.4. Distinguir variables Estáticas y Globales

Llegados a este punto, damos un paso atrás y volvemos a las variables, para distinguir entre variables estáticas (static) y globales (global). Las variables estáticas se definen dentro de una función, la primera vez que es llamada dicha función la variable se inicializa, guardando su valor para posteriores llamadas.

```
<?php
        function contador ()
        {
          static $count = 0;
          $count = $count + 1;
           return $count;
        }
        echo contador()."<BR>"; // imprimirá 1
        echo contador()."<BR>"; // imprimirá 2
        echo contador()."<BR>"; // imprimirá 3
?>
```

Las variables globales, no se pueden declarar dentro de una función, lo que hacemos el llamar a una variable que ya ha sido declarada, tomando el valor que tenga en ese momento, pudiendo ser modificado en la función.

```
<?php

        var $a = 1;
        function ver_a()
        {
          global $a;
          echo $a."<BR>"; // imprimirá el valor de $a
          $a += 1; // sumamos 1 a $a
        }

        echo ver_a(); // imprimirá 1
        echo ver_a(); // imprimirá 2
        $a = 7;
        echo ver_a(); // imprimirá 7
        echo ver_a(); // imprimirá 8

?>
```

5.5. Funciones Variables

PHP soporta el concepto de funciones variables, esto significa que si una variable tiene unos paréntesis añadidos al final, PHP buscará una función con el mismo nombre que el contenido de la variable, e intentará ejecutarla.

```
<?php

        function imprime($texto) {
          echo $texto . "\n";
        }

        function imprimeNegrilla($texto){
         echo "<B>$texto</B>\n";
        }

        $MiFunc = "imprime";
        $MiFunc("Hola");  //imprimirá Hola

        $MiFunc = "imprimeNegrilla";
        $MiFunc("Hola");  //imprimirá Hola

?>
```

5.6. Recursión

PHP también permite la recursión, es decir, una función se puede llamar así misma. Para aclarar el concepto de recursión, vamos a crear una función que comprueba si un número es entero o no.

Un número que no sea entero, tiene una parte entera y otra decimal (comprendida entre 0 y 1), lo que vamos a hacer para comprobar si un número es entero o no, será restarle 1 al número en cuestión hasta que nos quedemos sin parte entera, y entonces comprobaremos si tiene parte decimal (un poco lioso todo esto).

```
<?php
        function esEntero($numero) {
              if ($numero > 1) {
                 return (esEntero($numero -1));
              } elseif ($numero < 0) {
                /* como los núm. son simétricos
                chequeamos lo convertimos a positvo */
              return (esEntero((-1) * $numero -1));
              } elseif (($numero > 0) AND ($numero < 1)) {
              return ("NO");
              } else {
                 /* el cero es entero por definición */
                 return ("SI");
              }

        } //fin function

        echo "¿Es  0  un número entero? ".esEntero(0)."\n";
        echo "¿Es  3.5 un número entero? ".esEntero(3.5)."\n";
        echo "¿Es  -7  un número entero? ".esEntero(-7)."\n";
        echo "¿Es -9.2 un número entero? ".esEntero(9.2)."\n";
?>
```

5.7. Cómo ahorramos líneas de código

Por lo general, todos nuestros script tienen partes de código iguales, las funciones include() y require() nos van ahorrar muchas de estas líneas de código. Ambas funciones hacen una llamada a un determinado fichero pero de dos maneras diferentes, con include(), insertamos lo que contenga el fichero que llamemos de manera literal en nuestro script, mientras que con require(), le decimos que el script necesitará parte de código que se encuentra en el fichero que llama require().
Como todo esto es un poco lioso, veamos unos ejemplos que nos lo aclara.

```
<?php
include ("header.inc");
echo "Hola Mundo";
include ("footer.inc");
?>
```

Si tenemos encuenta que el fichero **header.inc** contiene:

```
<html>
<body>
```

y el fichero **footer.inc** contiene:

```
</body>
</html>
```

Nuestro script sería equivalente a:

```
<html>
<body>
<?php
        echo "Hola Mundo";
?>
</body>
</html>
```

Ahora veamos el script de ejemplo para la función require():

```
<?php
        require ("config.inc");
        include ("header.inc");
        echo $cadena;
        include ("footer.inc");
 ?>

        Donde el fichero config.inc tendría algo como ésto:
        <?php
          $cadena = "Hola Mundo";
        ?>
```

5.8. Funciones Tiempo y fecha

Vamos a ver como algunas funciones relacionadas con el tiempo y la fecha, así como algunos ejemplos prácticos.

-time

Devuelve el número de segundos transcurridos desde el 1 de Enero de 1970. A esta forma de expresar fecha y hora se le denomina **tmestamp.**

-date(formato, timestamp)

La función **date** devuelve una cte(formato, timestamp)
La función **date** devuelve una cadena formateada según los código de formato. Si no le pasamos la variable timestamp nos devuelve la cadena formateada para la fecha y la hora actual.

Los códigos de formato para la función date son:

CODIGO	DESCRIPCIÓN
a	am o pm
A	AM o PM
d	Día del mes con ceros
D	Abreviatura del día de la semana (inglés)
F	Nombre del mes (inglés)
h	Hora en formato 1-12
H	Hora en formato 0-23
i	Minutos
j	Día del mes sin ceros
l	Dia de la semana
m	Número de mes (1-12)
M	Abreviatura del mes (inglés)
s	Segundos
y	Año con 2 dígitos
Y	Año con 4 dígitos
z	Dia del año (1-365)

Para ver algunos ejemplos supongamos que ahora es el 7 de abril de 2000 a las 14 horas 30 minutos y 22 segundos:

- date("d-m-Y") -> 07-04-2000
- date("H:i:s") -> 14:30:22
- date("Y") -> 2000
- date("YmdHis") -> 20000407143022
- date("d/m/y H:i a") -> 07/04/00 14:30 pm
- date(d-m-Y H:i, time()) -> el momento actual

-mktime(hora, min, seg, mes, dia, año)

La funcion mktime devuelve una variable de tipo timestamp a partir de las coordenadas dadas. La principal utilidad de esta funcion es la de añadir o quitar una determinada cantidad de fecha u horas a una dada.

```
<?PHP
      function restarDias($numdias, $date) {
      if (isset($date)) {
         $date = time();
         }
      list($hora, $min, $seg, $dia, $mes, $anno) = explodest($hora, $min, $seg, $dia, $mes, $anno) = explode( " ",
      date( "H i s d m Y"));
      $d = $dia - $numdias;
      $fecha = date("d-m-Y", mktime($hora, $min, $seg, $mes, $d, $anno));
      return $fecha;
      }

      echo restarDias(5)."<BR>";
      echo restarDias(10)."<BR>";
?>
```

-chekdate (mes, dia, año)

La función chekdate comprueba si una fecha es válida, si es así devuelve TRUE y si no lo es FALSE. Una fecha se considera válida si el año está entre 1900 y 32767, el mes entre 1 y 12, y el dia es menor o igual que número de días total del mes en cuestión.

```
<?PHP
          if (checkdate(31, 2, 2000)) {
          echo "La fecha es correcta";
          } else {
          echo "La fecha es incorrecta";
          }
?>
```

Para el ejemplo anterior nos daría que la fecha es incorrecta, febrero nunca tiene un día 31.

Parte II :
Base de Datos MySQL con PHP

CAPITULO I : INTRODUCCION

1.1. Base de Datos de Ejemplo

Para comprender mejor como se programa en PHP para manipular datos a una Base de Datos MySQL, consideramos el siguiente ejemplo.

Base de Datos: AMIGOS
Tabla : AGENDA
Campos: Apellido, Nombre, Dirección, Teléfono y Email.

1.2. Registros (Datos) de Ejemplo

Apellido	Nombre	Dirección	Teléfono	Email

1.3. Crear la Base de Datos

El comando para crear una base de datos MySQL es el siguiente:

mysqladmin -u root create base_datos

Ejemplo**: mysqladmin -u** root **create** amigos

Con este comando conseguimos crear una base de datos en el servidor de bases de datos de nuestro servidor.

1.4. Crear Tablas – Varias Formas

a. Escribir ordenes directamente desde el prompt de MySql › Ejemplo: create table agenda

b. Escribir un fichero de texto con el contenido de la sentencia SQL equivalente y luego decirle al motor de base de datos que la ejecute con la siguiente instrucción:

mysql -u root amigos < **tablas.sql**

c. Una segunda forma de crear un fichero texto por ejemplo llamado **tablas.dump** y escribir ordenes SQL

create table agenda (……)\g
insert into agenda values (……)\g
.....

Ejecutar luego la orden **#cat tablas.dump | mysql amigos**

CAPITULO II : Funciones MySQL para Base de Datos.

2.1. Conectar y Cerrar una BD MySQL desde PHP

-**mysql_connect**("host", "usuario",”contraseña”);

Ejemplo: $link = **mysql_connect**("localhost", "nobody");

La función mysql_connect(), abre una conexión con el servidor **MySQL** en el **Host** especificado (en este caso la misma máquina en la que está alojada el servidor **MySQL = localhost**). También debemos especificar un usuario (**nobody**, **root**, etc.), y si fuera necesario un password para el usuario indicado. El resultado de la conexión es almacenado en la variable $link, en donde si toma el valor de 0 = No realizó Conexión y un número mayor a 0 = Conexión Ok.

-mysql_close(var.conexión)

Ejemplo: **mysql_close(**$link**)**

mysql_close es una función que cierra el enlace con la base de datos que anteriormente se ha abierto. No es necesaria su invocación, pues al finalizar el código php, se cerrarán todas las conexiones. Sin embargo, sí será necesaria si deseamos realizar dos conexiones a bases de datos distintas en la misma página php; primero habrá que cerrar la que tengamos abierta y luego volver a conectar.

2.2. Seleccionar la Base de Datos

-**mysql_select_db**("base de datos", var_conexión);

Ejemplo: **mysql_select_db**("amigos", $link);

Con mysql_select_db(), **PHP** le dice al servidor que en la conexión $link nos queremos conectar a la base de datos **amigos**.

2.3. Consultar en Instrucciones Lenguaje SQL

-**mysql_query**("Instrucción SQL", var_conexión);

Ejemplo: $result = **mysql_query**("SELECT * FROM agenda", $link);

La siguiente función mysql_query(), es la que hace el trabajo duro, usando el identificador de la conexión ($link), envía una instrucción **SQL** al servidor **MySQL** para que éste la procese. El resultado de ésta operación es almacenado en la variable $result, si toma un valor de 0 = sentencia incorrecta y un valor mayor a 0 = Consulta satisfactoria.

2.4. Mostrar Datos de los Campos

-**mysql_result**(var_resultado, pos_reg, campo);

Ejemplo: **mysql_result**($result, 0, "nombre");

mysql_result() es usado para mostrar los valores de los campos devueltos por la consulta ($result). En este ejemplo mostramos los valores del **registro 0**, que es el primer registro, y mostramos el valor de los campos especificados.

2.5. Ejemplo 01 de Consulta a una BD

En este ejemplo, se consulta el primer registro (posición 0) de la tabla Agenda de la Base de Datos Amigos.

```
<html>
```

```
<body>

<?php
   $link = mysql_connect("localhost", "nobody");
   mysql_select_db("amigos", $link);
   $result = mysql_query("SELECT * FROM agenda", $link);
   echo "Apellido: ".mysql_result($result, 0, "apellido")."<br>";
   echo "Nombre: ".mysql_result($result, 0, "nombre")."<br>";
   echo "Dirección: ".mysql_result($result, 0, "direccion")."<br>";
   echo "Teléfono :".mysql_result($result, 0, "telefono")."<br>";
   echo "E-Mail :".mysql_result($result, 0, "email")."<br>";
?>

</body>
</html>
```

2.6. Obtener un Registro en un Array de Datos

- mysql_fetch_row(var_resultado);

Ejemplo: $reg = mysql_fetch_row($result))

mysql_fetch_row(), devuelve un array con el contenido del registro actual (que se almacena en $reg) y avanza una posición en la lista de registros devueltos en la consulta **SQL**. Pero cada campo se referencia por un número $reg[0].

- **mysql_fetch_array**(var_resultado);

Ejemplo: $reg = **mysql_fetch_array**($result);

mysql_fetch_array() hace exactamente lo mismo que mysql_fetch_row(), con la excepción que podemos referenciar a los campos por su nombre ($reg["email"]), en vez de por un número.

2.7. Ejemplo 02 de Consulta a una BD

```
<html>
<body>
   <?php
        $link = mysql_connect("localhost", "nobody");
        mysql_select_db("amigos", $link);

        $result = mysql_query("SELECT nombre, email FROM agenda", $link);

        if ($row = mysql_fetch_array($result)){
           echo "<table border = '1'> \n";
           echo "<tr><td>Nombre</td><td>E-Mail</td></tr> \n";

           do {
              echo "<tr><td>".$row["nombre"]."</td><td>".$row["email"]."</td></tr> \n";
           } while ($row = mysql_fetch_array($result));

           echo "</table> \n";
        } else {
          echo "i No se ha encontrado ningún registro !";
        }
   ?>
  </body>
</html>
```

Con la sentencia if/else, asignamos a $row el primer registro de la consulta, y en caso de no haber ninguno (else) mostramos un mensaje (**"No se ha encontrado..."**). Mientras que con la sentencia do/while, nos aseguramos que se nos muestren todos los registros devueltos por la consulta en caso de haber más de uno.

Hay que destacar la utilización del **punto (.)**, como operador para concatenar cadenas.

CAPITULO III : Manipulación de Datos con Sentencias SQL

3.1. Modificar registros

UPDATE tabla SET campo1=valor1, campo2=valor2,... WHERE condiciones;

Para modificar hay que tener permiso para ello en el servidor de BD, el resto nos viene de corrido. Primero seleccionamos el registro que deseamos modificar, y luego, mandamos una consulta con las modificaciones, o ambas cosas a la vez. Suponemos que las modificaciones las recogemos de un formulario.

```
<html>
<body>
   <?php
        if (isset($id)){
          $link = mysql_connect("localhost", "root");
          mysql_select_db("amigos",$db);

          $sql = "UPDATE agenda SET apellido='$apellido', nombre='$nombre', direccion='$direccion',".
                                   "telefono='$telefono', email='$email' WHERE id=$id";
          $result = mysql_query($sql);

        }else{
          echo "Debe especificar un 'id'.\n";
        }
   ?>
</body>
</html>
```

3.2. Borrar registros

DELETE FROM tabla WHERE condiciones;

El proceso de borrar un registro es idéntico al de modificar, solo que en vez de utilizar UPDATE utilizamos DELETE.

```
<html>
<body>

   <?php
        if (isset($id)){
          $link = mysql_connect("localhost", "root");
          mysql_select_db("amigos",$db);

          $sql = "DELETE agenda WHERE id=$id")
          $result = mysql_query($sql);

        }else{
          echo "Debe especificar un 'id'.\n";
        }
   ?>
</body>
</html>
```

3.3. Añadir registros a nuestra base de datos

Para insertar datos se sigue siempre el mismo procedimiento, una sentencia SQL del tipo:

INSERT INTO tabla (campo1,campo2,campo3,...) VALUES (valorcampo1 ,valorcampo2 , valorcampo3, ...);

Nota Importante: Las cadenas de texto ó lo que se entiende por "VARCHAR ó TEXT" deben de ir entre comillas. Los números no tienen porque ir entre comillas.

```
<html>
<body>

   <form method="post" action="add_reg.php">

        Apellido   :<input type="Text" name="apellido"><br>
        Nombre   :<input type="Text" name="nombre"><br>
        Dirección:<input type="Text" name="direccion"><br>
        Teléfono :<input type="Text" name="telefono"><br>
        E-mail   :<input type="Text" name="email"><br>
        <input type="Submit" name="enviar" value="Aceptar información">

   </form>

</body>
</html>
```

Hemos creado un formulario donde recoger los datos, y una vez introducidos ejecutamos un **script** llamado **add_reg.php**, pues veamos como es este **script**.

```
<html>
<body>

   <?php

        $link = mysql_connect("localhost", "root");

        mysql_select_db("amigos",$link);

        $sql = "INSERT INTO agenda (apellido, nombre, direccion, telefono, email) " .
          "VALUES (`$apellido', '$nombre', '$direccion', '$telefono', '$email')";

        $result = mysql_query($sql);

        echo "¡Gracias! Hemos recibido sus datos.\n";

   ?>
</body>
</html>
```

Como se puede ver, para introducir un nuevo registro, utilizamos la ya conocida función mysql_query(), la cúal también usamos para las consultas, y usaremos para las actualizaciones, es decir una señora **función**. ¡Aaah!, una cosa muy importante, para poder añadir o modificar registros debemos tener permiso para ello en el servidor **MySQL**, por eso en este caso me conecto como **root**, pero podría ser cualquier otro usuario.

Un ejemplo más, vamos a combinar la página web de formulario y el fichero de script **php**, en un solo fichero que llamaremos **add_reg.php.**

```
<html>
<body>

   <?php
        if ($enviar) {
```

```
        $link = mysql_connect("localhost", "root");
        mysql_select_db("amigos",$link);

       $sql = "INSERT INTO agenda (apellido, nombre, direccion, telefono, email) " .
                    "VALUES ('$apellido', '$nombre', '$direccion', '$telefono', '$email')";

        $result = mysql_query($sql);
        echo "¡Gracias! Hemos recibido sus datos.\n";

      }else{
      ?>

        <form method="post" action="add_reg.php">
              Apellido   :<input type="Text" name="apellido"><br>
              Nombre   :<input type="Text" name="nombre"><br>
              Dirección:<input type="Text" name="direccion"><br>
              Teléfono :<input type="Text" name="telefono"><br>
              E-mail   :<input type="Text" name="email"><br>
              <input type="Submit" name="enviar" value="Aceptar información">
        </form>

      <?php
      } //end if
  ?>

</body>
</html>
```

3.4. Funciones mysql_errno y mysql_error

Cuando mandamos una sentencia SQL al servidor, ésta puede dar una respuesta correcta o una respuesta incorrecta. Estas sentencias nos dicen que nº de error se ha producido (mysql_errno) y el texto de error que el servidor MySQL ha respondido. En este caso pueden ocurrir varias cosas, que algún registro esté mal formateado (por ejemplo, que el e-mail no sea un número) ó que ya haya otro registro con el nombre especificado. Ambos casos tendrán un número de error diferente y un mensaje de error diferente.

CAPITULO IV : Aplicaciones Diversas - Ejemplos

4.1. Un buscador para nuestra base de datos

Vamos a ver una aplicación, un ejemplo, de todo lo visto hasta ahora. Escribiremos un **script** que sirva para buscar una determinada cadena (que recibiremos de un formulario, y la almacenamos en la variable $buscar), dentro de nuestra base de datos, concretamente dentro del **campo** "nombre".

En primer lugar escribiremos el texto **HTML** de la página web que nos servirá como formulario de entrada, la llamaremos **formulario.htm**.

```
<html>
<body>
   <form method="POST" action="http://localhost/buscador.php">
        <strong>Palabra clave:</strong> <input type="text" name="buscar" size="20"><br><br>
        <input type="submit" value="Search" name="enviar">
   </form>
</body>
</html>
```

El siguiente **script** de búsqueda lo llamaremos **buscador.php**, y será el encargado de hacer la búsqueda en la BD, y devolver por pantalla los registros encontrados.

```
<html>
<body>

   <?php
   if (!isset($buscar)){
        echo "Debe especificar una cadena a buscar";
        echo "</html></body> \n";
        exit;
   }
   $link = mysql_connect("localhost", "nobody");
   mysql_select_db("amigos", $link);

   $result = mysql_query("SELECT * FROM agenda WHERE nombre LIKE '%$buscar%' ORDER BY nombre", $link);

   if ($row = mysql_fetch_array($result)){
              echo "<table border = '1'> \n";
        //Mostramos los nombres de las tablas
        echo "<tr> \n";

        while ($field = mysql_fetch_field($result)){
                  echo "<td>$field->name</td> \n";
        }
       echo "</tr> \n";

   do {
             echo "<tr> \n";
             echo "<td>".$row["id"]."</td> \n";
             echo "<td>".$row["nombre"]."</td> \n";
             echo "<td>".$row["direccion"]."</td> \n";
             echo "<td>".$row["telefono"]."</td> \n";
             echo "<td><a href='mailto:".$row["email"]."'>".$row["email"]."</a></td> \n";
             echo "</tr> \n";
       } while ($row = mysql_fetch_array($result));
             echo "</table> \n";
   } else {
   echo "¡ No se ha encontrado ningún registro !";
   }
   ?>
```

```
</body>
</html>
```

Lo primero que comprobamos es que el contenido de la variable $buscar que recibimos del la página web **formulario.htm** no es una cadena vacía, y esto lo hacemos con la función isset() que devuelve **'falso'** si la variable que recibe está vacía. A la función le anteponemos el signo admiración (!) que es equivalente a un **NOT**, para convertirlo en **'verdadero'** en caso de que la variable esté vacía, y en es caso terminamos la ejecución del **script** con exit.

Lo más importante de este **script**, es sin duda la sentencia **SQL** que le enviamos al servidor **MySQL**, y más concretamente la condición que le imponemos, WHERE nombre LIKE '%$buscar%'. Con la sentencia LIKE buscamos cualquier ocurrencia de la cadena contenida en $buscar, mientras que con los signos de porcentaje (%) indicamos el lugar de la coincidencia, por ejemplo, si hubiésemos puesto nombre LIKE '%$buscar', buscaríamos cualquier ocurrencia al final del **campo** "nombre", mientras que si hubiésemos puesto nombre LIKE '$buscar%', buscaríamos cualquier ocurrencia al principio del **campo** "nombre".

La última novedad que hemos incorporado, es la función mysql_fetch_field, con el que obtenemos información acerca de las características de cada **campo**, como su nombre, tipo, longitud, nombre de la tabla que los contiene, etc.

4.2. Envió de emails

PHP nos ofrece la posibilidad de enviar emails de una manera sencilla y fácil, para ello el lenguaje nos proporciona la instrucción mail( )

```
<?php
  mail(destinatario, tema, texto del mensaje);
?>
```

En el parámetro destinatario pondremos la dirección de email a donde se enviará el mensaje, en el parámetro tema el tema o subject del mensaje y el parámetro texto del mensaje el cuerpo del mensaje en formato texto plano.

Existe una sintaxis extendida de la instrucción mail( ) que nos permite añadir información adicional a la cabecera del mensaje.

```
<?php
  mail(destinatario, tema, texto del mensaje, información adicional de cabecera);
?>
```

En la información de cabecera podremos incluir parámetros adicionales al mensaje como Reply-To:, From:, Content-type:... que nos permiten tener un mayor control sobre el mensaje.

```
<html>
<body>

<H1>Ejemplo de envio de email</H1>
Introduzca su direccion de email:

<FORM ACTION="email.php" METHOD="GET">
                <INPUT TYPE="text" NAME="direccion"><BR><BR>
                Formato: <BR>
                <INPUT TYPE="radio" NAME="tipo" VALUE="plano" CHECKED> Texto plano<BR>
                <INPUT TYPE="radio" NAME="tipo" VALUE="html"> HTML<BR><BR>
                <INPUT TYPE="submit" VALUE="Enviar">
</FORM>

</body>
</html>
```

Código del fichero **email.php**

```
<html>
<body>

<H1>Ejemplo de envio de email</H1>

<? if ($direccion!=""){
  if ($tipo=="plano"){

    // Envio en formato texto plano

    mail($direccion,"Ejemplo de envio de email","Ejemplo de envio de email de texto plano\n\n
          http://localhost/\n Saludos desde Chimbote.\n");
  } else {

    // Envio en formato HTML

    mail($direccion,"Ejemplo de envio de email","<html><head></head>
       <body>Ejemplo de envio de email de HTML<br><br>http://localhost/<br>
         <u>Saludos</u> desde <b>Chimbote</b></body></html>","Content-type: text/html\n");
  }
 echo "Se ha enviado un email a la direccion: ",$direccion," en formato <b>",$tipo,"</b>.";
}
?>


</body>
</html>
```