PHP

1. Instalación de PHP

En este texto vamos a ver una introducción a PHP. Usaremos PHP para crear sitios web dinámicos. Por tanto, necesitaremos tener acceso a un servidor web que tenga soporte de PHP. Existen varias alternativas:

- Tener contratado alojamiento Web con un servidor que permita el uso de PHP. Así, las páginas que vayamos creando podremos publicarlas en nuestro servidor y comprobar que se comportan correctamente. Es una opción muy fiable, pero lenta, porque cada nuevo cambio habría que "subirlo" al servidor para poder probar su resultado.
- Instalar un servidor web en nuestro ordenador, y así poder probar rápidamente. Así sólo subiríamos a nuestro servidor externo (si es que lo tenemos) las páginas cuando estén terminadas. Aquí nos encontramos nuevamente con dos opciones mayoritarias:
    - Usar un "entorno LAMP": un sistema operativo Linux, con el servidor web Apache, la base de datos MySQL y el entorno de PHP. Es lo recomendable, pero exige ser un usuario relativamente avanzado, porque quizá la configuración del sistema no sea trivial.
    - Usar un sistema operativo Windows/Linux y descargar un entorno que incluya Apache + MySQL + PHP listo para usar paquetes "WAMP/XAMPP" que suelen ser muy sencillos de poner en funcionamiento: incluyen una versión instalable o un fichero ZIP que descomprimir, e incluso la posibilidad de instalar como "servicio" (de modo que siempre esté activo) o como "programa" (para que esté activo sólo cuando nosotros lo pidamos, y no consuma recursos el resto del tiempo). En muchas ocasiones incluso son versiones "portables", que pueden funcionar desde un pendrive o un disco externo. Nosotros usaremos esta opción con una de las versiones de Xampp.

Una vez instalado el Xampp, lanzamos todos los servicios y veremos que nuestro entorno de prueba está listo. Para probarlo, deberemos entrar a nuestro navegador, y como dirección, poner la de nuestro propio ordenador, que es "***http://localhost/***". Nos aparecera el ***index.php*** de la carpeta Dashboard porque el index.php nos direcciona a ella. Para que no aparezca esto y podamos trabajar como si de un servidor ftp fuese, entramos dentro de la carpeta ***/opt/lampp/htdocs*** y renombramos el archivo ***index.php*** a ***index_old.php***. Una vez realizado esto crearemos una carpeta llamada ejercicios dentro de htdocs y le daremos permisos de escritura y lectura a todos. Para parar todos los servicios haríamos mediante terminal: ***sudo /opt/lampp/lampp stop*** y para iniciarlo ***sudo /opt/lampp/lampp start***

Si nuestro servidor está mostrando páginas correctamente, llega el momento de crear nuestra propia página de prueba en PHP. Por ahora sólo veremos una orden de PHP, que será la que escriba algo en la ventana del navegador:

Para escribir un texto se usa la orden "echo" , seguido del texto entre comillas, y se termina con un punto y coma, así:

```
echo"Hola!";
```

Y esta orden (y las demás que iremos viendo de PHP), las escribiremos dentro de un fichero HTML "casi normal", apenas con un par de diferencias:

- Las órdenes en PHP se deben indicar entre <?php y ?>
- El fichero se debe guardar con el nombre que queramos, pero terminado con ".php".

Por tanto, para crear una primera página que simplemente nos salude, podríamos hacer lo siguiente:

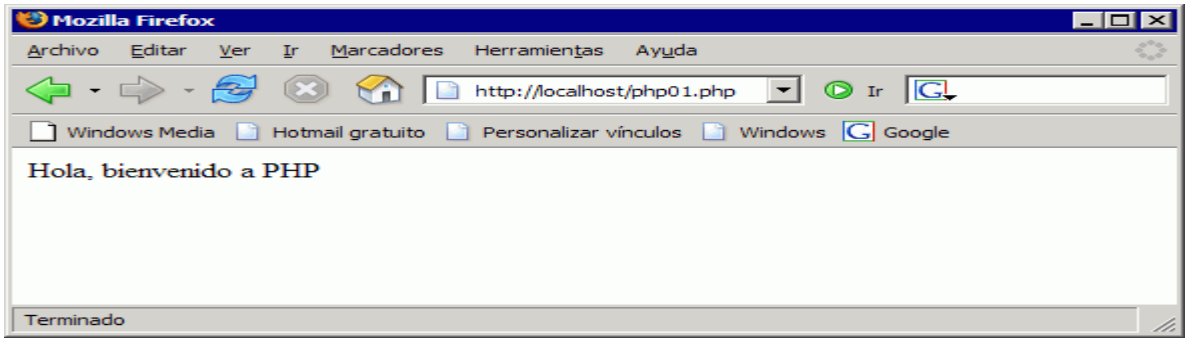
```
<html>
<body>
<?php
echo "Hola, bienvenido a PHP";
?>
</body>
</html>
```

¿Y esto donde lo guardamos? En la carpeta en la que estuviera instalado nuestro servidor (en nuestro caso, "***/opt/lampp/htdocs/ejercicios***"), y con un nombre que termine con ".php", por ejemplo "ejemplo1.php". En otros paquetes LAMP, la carpeta no se llama "htdocs" sino "www".

Para comprobar cómo ha quedado, le decimos a nuestro navegador que nos muestre la página "php01.php" de nuestro ordenador ("localhost"), escribiendo en la barra de direcciones

http://localhost/ejemplo1.php

El resultado sería este:



Un último detalle importante: PHP es un lenguaje de servidor. El "cliente" que ve la página en su ordenador, no podrá ver qué órdenes de PHP hemos usado, sino su resultado. Si le pedimos a nuestro navegador (Internet Explorer, Firefox, Chrome, Safari, Opera o el que sea) que nos muestre el "código fuente" de la primera página que hemos creado con PHP, vería algo como esto

```
<html>
<body>
Hola, bienvenido a PHP
</body>
</html
```

Como se puede observar, no aparece por ningún sitio la orden "echo", sólo el resultado de esa orden.

1.1. Caracteres especiales

Un ***string*** o **cadena de caracteres** es una serie de caracteres donde cada carácter equivale a un ***byte***. Esto conlleva a que PHP sólo admite un conjunto de 255 caracteres, y por tanto no hay **soporte nativo** para Unicode.

El **tamaño máximo de un *string*** son 2GB (2147483647 bytes).

Existen varias formas de mostrar un *string*:

1. Entrecomillado simple
2. Entrecomillado doble
3. Análisis de variables

A. Entrecomillado simple

Es la forma más sencilla de delimitar un *string*: '*string*'.

Si se quiere especificar una **comilla simple dentro de un *string***, se escapa con una barra invertida \, y se hace lo mismo para especificar una **barra invertida dentro de un *string*** \. Cualquier otra secuencia de escape o variable se mostrará literalmente, por ejemplo: **\r**, **\n**, **$variable**...

Ejemplos:

```
<?php
echo 'Cadena simple';

echo 'Las nuevas líneas
pueden formarse de
esta forma
';

echo 'En inglés "do not" se puede escribir como "don\'t"';

echo 'En Windows el directorio base es C:\\';

echo 'No funcionan ni las variables como $variable ni
otras sentencias de escape para caracteres especiales como \n';
```

B. Entrecomillado doble

Si el string está delimitado con **comillas dobles** "*string*", **PHP** interpretará más sentencias de escape para caracteres especiales:

Secuencia	Significado
\n	avance de línea
\r	retorno de carro
\t	tabulador horizontal
\v	tabulador vertical
\e	escape

\f	avance de página
\	barra invertida
\$	signo de dólar
\"	comillas dobles
[0-7]{1,3}	secuencia de caracteres que coincida con la expresión regular, carácter en notación octal
\x[0-9A-Fa-f]{1,2}	secuencia de caracteres que coincida con la expresión regular, carácter en notación hexadecimal

Lo más significativo del entrecomillado doble de un string es el hecho de que **se expanden los nombres de las variables**.

C. Análisis de variables

Si un *string* se especifica con **comillas dobles** o **heredoc**, las variables que se nombren en su interior se analizarán. Existen dos tipos de sintaxis: **simple** y **compleja**. En la **compleja** los elementos van delimitados por llaves.

C.1. Sintaxis simple

La sintaxis **simple** es la más empleada en la práctica y ofrece una forma sencilla de embeber una variable, un valor de array o una propiedad de un objeto:

```
$animal = "perro";
echo "Yo tengo un $animal" . PHP_EOL;
```

```
$animales = ['perro', 'gato', 'jabali', 'insecto'=>'cucaracha'];
echo "Yo tengo un $animales[1]" . PHP_EOL;
echo "Yo tengo un $animales[0]" . PHP_EOL;
echo "Yo quiero tener una $animales[insecto] como mascota" . PHP_EOL;
// Nótese que insecto no lleva comillas
```

```
class Usuario {
   public $carlos = "Carlos Perez";
   public $sonia = "Sonia Fernandez";
}
$usuario = new Usuario();

echo "El usuario $usuario->carlos tiene un $animales[0]" . PHP_EOL;
echo "El usuario $usuario->sonia tiene 2 $animales[1]s" . PHP_EOL;
```

C.2. Sintaxis compleja

La sintaxis compleja permite **expresiones complejas**. Cualquier **variable**, **array** o **propiedad de objeto** con una representación *string* se puede incluir con esta sintaxis. Se escribe de la misma forma que la anterior pero delimitándola con { y }. Los corchetes no pueden ser escapados, por lo que se reconoce sólo cuando le siga un símbolo del dólar $ (sin espacio de por medio). Para un {$ literal se utiliza {\$. Ejemplos:

- **Acceso a una simple variable**

```
$color = "verde";
echo "Ese color es { $color}" . PHP_EOL; // Muestra: { verde}
echo "Ese color es {$color}" . PHP_EOL; // Muestra: verde
echo "Ese color es ${color}" . PHP_EOL; // Muestra: verde
```

- **Acceso a propiedades de clases**

```
class Coche {
  public $color = "azul";
  public $tipo = array("Marca" => "SEAT", "Modelo" => "Toledo");
}
$miCoche = new Coche;
```

Para acceder a un valor de propiedad puede usarse sintaxis compleja o simple:

```
echo "Ese coche es {$miCoche->color}" . PHP_EOL;
echo "Ese coche es $miCoche->color" . PHP_EOL;
```

Si se trata de valores de array dentro de propiedades, necesitamos la compleja:

```
echo "Los coches {$miCoche->tipo['Marca']} son de buena calidad" . PHP_EOL;
```

Funciona sin comillas, pero genera un Notice:

```
echo "Marca de coche: {$miCoche->tipo[Marca]}" . PHP_EOL; // Notice "Undefined constant"
```

- **Acceso a array multidimensional**

```
$coche = array(
    "Marca" => array (
        0 => "Toyota",
        1 => "Mercedes"
    )
);
```

De nuevo accedemos con comillas si el array es asociativo:

```
echo "La marca de mi coche es: {$coche["Marca"][0]}" . PHP_EOL;
```

También es posible con arrays multidimensionales en propiedades:

```
echo "String a mostrar: {$objeto->variables[2]->nombre}" . PHP_EOL;
```

- **Variables variables**

```
$animal = "perro";
$perro ="Bobby";
echo "Nombre de mi perro: {${$animal}}" . PHP_EOL; // Devuelve: Nombre de mi perro: Bobby
```

2. Variables en PHP

Un primer paso para ganar en versatilidad es no escribir siempre textos prefijados. Si lo que escribimos se indica entre comillas, aparecerá tal cual; si no está entre comillas, el intérprete de PHP intentará deducir cual es su valor.

Así, nuestro segundo ejemplo (ejemplo2.php) podría ser un sencillo programa que sumara dos números, así

```
<html>
<body>
<?php
// ejemplo2.php - Valores calculados

echo "2+3 es ";
echo 2+3;
echo " ¿verdad?";
?>
</body>
</html>
```

Su resultado sería este:

```
2+3 es 5 ¿verdad?
```

La línea que comienza por doble barra es un comentario, que puede ser útil para el programador pero no afectará al resultado del programa.

Podemos escribir varias cosas con la orden "echo"; basta separarlas con comas).

```
<html>
<body>
<?php
// php02b.php - Valores calculados y "echo" con coma
echo "2+3 es ", 2+3, " ¿verdad?";
?>
</body>
</html>
```

que tendría el mismo resultado:

```
2+3 es 5 ¿verdad?
```

En la práctica, tampoco es normal que esos valores a calcular estén prefijados, sino que estén almacenados en una **variable**, porque procedan de un cálculo posterior. Dentro de poco veremos cómo hacer cálculos sencillos, y más adelante veremos cómo recibir datos desde otra página. De momento, nuestras variables tendrán valores prefijados por nosotros.

Para definir una variable deberemos usar un nombre que comience por el símbolo $. Al contrario que en otros muchos lenguajes, en PHP no es necesario declarar las variables antes de usarlas, ni indicar qué tipo de datos van a almacenar (se deducirá del contexto).

Nuestro tercer ejemplo (ejemplo3.php), que maneje alguna variable, podría ser

```
<html>
<body>
<?php
// ejemplo3.php - Variables

$nombre = "clase";
$numero = 5;
$segundoNumero = 6.7;
$negritaSI = "<b>";
$negritaNO = "</b>";

echo "Hola, ", $negritaSI, $nombre, $negritaNO,
"<br />El primer numero es ", $numero,
" y el segundo es ", $segundoNumero;
?>
</body>
</html>
```

Que en el navegador se vería así:

Hola, **clase**
El primer numero es 5 y el segundo es 6.7

Como se ve en este ejemplo, se pueden escribir códigos HTML dentro de una orden "echo", bien sea directamente, o con la ayuda de variables.

Si pedimos operaciones numéricas, como sumar o multiplicar, los datos se convierten a numéricos automáticamente (si es posible). Por ejemplo, si sumamos una variable de valor 10 y otra de valor "7" (una cadena), el resultado será 17. Si lo que queremos es concatenar dos cadenas de texto (crear una nueva cadena formada por la unión de esas dos), no usaremos el operador "+", sino el punto "."; vamos a ver las operaciones más habituales en un ejemplo (php04):

```
<html>
<body>
<?php
// ejemplo4.php - Variables y operaciones aritméticas

$n1 = 201;
$n2 = "35";

echo "Los numeros son ", $n1, " y ", $n2, ".<br />";
echo "Suma: ", $n1+$n2,"<br />";
echo "Resta: ", $n1\-$n2,"<br />";
echo "Producto: ", $n1\*$n2,"<br />";
echo "Division: ", $n1/$n2,"<br />";
printf("Division sin decimales: %d<br />", $n1/$n2);
echo "O bien: ", intval($n1/$n2), "<br />";
echo "Resto de la division: ", $n1 % $n2, "<br />";
echo "Concatenados: ", $n1 .$n2, "<br />";
?>
</body>
</html>
```

que daría como resultado

```
Los numeros son 201 y 35.
Suma: 236
Resta: 166
Producto: 7035
Division: 5.7428571428571
Division sin decimales: 5
O bien: 5
Resto de la division: 26
Concatenados: 20135
```

Hay varias cosas que comentar en este programa:

- La variable $n2 contiene una cadena de texto, pero aun así se puede operar con ella como si fuera un número, porque el sistema convierte de cadena a dato numérico de forma transparente.
- Las operaciones matemáticas son las habituales en la mayoría de lenguajes de programación: + - / *. También se puede calcular el resto de la división con %.
- Al contrario que en otros lenguajes como C y C#, la división siempre da un resultado con decimales, aunque los dos operandos sean enteros
- Si no queremos mostrar los decimales, podemos usar "**printf**", en vez de "**echo**". El código de formato "%d" hará que se muestre como un número entero. Una alternativa es usar "intval", que nos da el valor entero del dato que se indique entre paréntesis.
- Podemos concatenar dos cadenas de texto, para obtener una nueva cadena a partir de ellas, usando el operador "."

Existe una **sintaxis abreviada** para "echo", que consiste en forma todavía más sencilla de hacerlo, que es usando sólo un símbolo de "igual" (=). En versiones de PHP anteriores a la 5.4.0 puede no estar disponible, según sea la configuración del sistema (veremos más adelante cómo cambiarla). Por ejemplo, ésta sería una forma alternativa para el fichero ejemplo2.php, que también escribiría 5 en pantalla:

```
<html>
<body>
<?=2+3?>
</body>
</html>
```

En PHP se pueden usar tanto las comillas dobles como las simples, pero el comportamiento no es el mismo: si incluimos el nombre de una variable dentro de unas comillas dobles, se reemplazará por su valor, pero no si son comillas simples:

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

// ejemplo5.php - Comillas simples y dobles

$numero = 201;

echo "Con comillas dobles: $numero <br />";
echo 'Con comillas simples: $numero <br />';
?>
</body>
</html>
```

que mostraría:

```
Con comillas dobles: 201
Con comillas simples: $numero
```

3. Formularios. Comunicación entre páginas

Casi cualquier programa "real" necesita tener datos de entrada que manipular. Más adelante veremos cómo tomarlos a partir de las fecha y la hora del sistema, de una base de datos, de un fichero, de otra página web... pero por ahora comenzaremos pidiendo esos datos al usuario a través de un **formulario**.

Un formulario básico en HTML tendría esta apariencia (fichero "envia.html"):

```
<html>
<body>
  <form action="saludador.php" method="get">
    Introduce tu nombre: <input name="nombre" type="text"> <br /> <br />
    <input value="Saludar" type="submit"><br />
  </form>
</body>
</html>
```

Que en pantalla se vería algo así:

```
Introduce tu nombre:
```

Es decir, contiene un texto normal, una casilla de introducción de texto (tipo "text", llamada "nombre") y un botón (tipo "submit", que muestra el texto "Saludar"). Cuando se hace clic, se salta a una página llamada "saludador.php", y se le pasan los datos que hayamos indicado en el formulario (en nuestro caso, el nombre).

Estamos usando la forma de comunicación conocida como **"get"**, lo que se refleja en que los datos que pasamos de una página a otra se pueden leer en la barra de direcciones:

saludador.php?nombre=Juan

Nuestro "saludador" podría ser algo así:

```
<html>
<body>
Bienvenido!<br />
<?php
  echo "Hola " . $_GET["nombre"] . ", Como estas?";
?>
</body>
</html>
```

O también podría ser algo así:

```
<html>
<body>
Bienvenido!<br />
<?php
  echo "Hola " . $_REQUEST["nombre"] . ", Como
estas?";
?>
</body>
</html>
```

O también:

```
<html>
<body>
Bienvenido!<br />
$nombre=$_REQUEST[“nombre]”;  // o con $_GET
<?php
  echo "Hola " . $nombre . ", Como estas?";
?>
</body>
</html>
```

De modo que en pantalla obtendríamos

```
Bienvenido!
Hola Juan, Como estas?
```

Es decir, desde PHP leemos los datos pasados usando **$_GET** e indicando entre corchetes y comillas el nombre del campo correspondiente.

Vamos a mejorarlo un poco. Una página que leyera **dos números** para sumarlos después podría ser así (sumar.html):

```
<html>
<body>
  <form action="sumador.php" method="get">
    Primer numero: <input name="n1" type="text"> <br /> <br />
    Segundo numero: <input name="n2" type="text"> <br /> <br />
    <input value="Sumar" type="submit"><br />
   </form>
</body>
</html>
```

Y el programa PHP encargado de recibir los datos y sumar podría ser éste:

```
<html>
<body>
<?php
  $primerNumero = $_GET["n1"];
  $segundoNumero = $_GET["n2"];
  echo "$primerNumero + $segundoNumero es ";
  echo $primerNumero+$segundoNumero;
?>
</body>
</html>
```

Si el usuario teclea los números 25 y 31, en la barra de direcciones del navegador se podría leer:

sumador.php?n1=25&n2=31

Y en pantalla se vería

```
25 + 31 es 56
```

Esta forma de pasar parámetros es cómoda cuando el usuario realiza búsquedas, por ejemplo, porque puede modificar los datos en la propia barra de direcciones, y hacer una nueva búsqueda sin necesidad de volver a rellenar el formulario. Pero es totalmente **inapropiada** si hay datos que no

deban ser visibles, como una **contraseña**. Enseguida veremos una alternativa que es útil para estos casos.

Una alternativa que nos permite pasar datos sin que se vean en la barra del navegador es el **método POST**, así como más elementos que pueden aparecer en los formularios: casillas para contraseñas, recuadros para escribir varias líneas, listas desplegables y "botones de radio".

Vamos a hacerlo directamente con un ejemplo. Nuestro nuevo formulario (envia2.html) podría ser así:

```
<html>
<body>
 <form action="saludador2.php" method="post">
  Introduce tu nombre: <input name="nombre" type="text"> <br />
  Y tu contrase&ntilde;a: <input name="clave" type="password"> <br />
  Prefieres que te diga...<br />
    <input name="tipoSaludo" value="hola"
checked="checked" type="radio">Hola<br />
    <input name="tipoSaludo" value="adios"
type="radio">Adios<br />
  Opci&oacute;n extra: <select name="posibilidad" size="1">
    <option value="opcion1">Primera posibilidad </option>
    <option value="opcion2">Segunda posibilidad </option>
    <option value="opcion3">Tercera posibilidad </option>
   </select> <br />
  Comentario: <br />
    <textarea rows="10" name="texto" cols="40"></textarea>
  <br />
  <input value="Saludar otra vez" type="submit"><br />
 </form>
</body>
</html>
```

En este ejemplo, hemos usado:

- Un campo "password" para la contraseña (el valor introducido no se verá en pantalla).
- Varias opciones de tipo "radio", excluyentes (sólo se puede escoger una de ellas).
- Una lista "select", que tiene tamaño (size) 1, por lo que las demás opciones se despliegan, en vez de verse continuamente en pantalla.
- Un área de introducción de varias líneas de texto, "textarea", en concreto pensado para mostrar 10 filas y 40 columnas.

Podríamos acceder a los datos con un fuente en PHP como este

```
<html>
<body>
Bienvenido!<br />
<?php
 echo "Hola " . $_POST["nombre"] . ", Como estas?";
 echo "<br />Tu contraseña era " . $_POST["clave"];
 echo "<br />Querías que te dijera " . $_POST["tipoSaludo"];
 echo "<br />Y en la lista has elegido " . $_POST["posibilidad"];
 echo "<br />Finalmente, en el area de texto has escrito:<br />" . $_POST["texto"];
?>
</body>
</html>
```

Así, si rellenamos el formulario anterior al visualizarlo en el navegador, podríamos obtener el siguiente resultado:

```
Bienvenido!
Hola juan, Como estas?
Tu contraseña era 1234
Querías que te dijera adios Y en la lista has elegido opcion2
Finalmente, en el area de texto has escrito: Esta es una frase de ejemplo
```

Un último detalle algo más avanzado: si queremos que **una página se llame a sí misma**, en vez de tener una página con el formulario y otra página que recibe los datos, no necesitamos hacer grandes cambios:

- La página deberá tener un "if" ("si", condicional) que compruebe el valor de una variable recibida por POST (o por GET) y según eso, muestre el formulario o analice los datos. En el próximo apartado veremos los detalles de la orden "if".
- En el "action" del formulario deberemos poner el nombre de la misma página que muestra el formulario.

Puede ser preferible usar "$_SERVER['PHP_SELF']" en vez del nombre de la página, en el "action", de modo que se siga comportando correctamente aunque cambiemos el nombre del fichero (de lo contrario, podemos olvidar cambiarlo en el "action" y dejar la página "huérfana").

Un ejemplo de formulario (saludar.php) que se llame a sí mismo para saludar a la persona que haya introducido su nombre sería así:

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

        if ( ! isset($_POST["nombre"]) )
        {
?>
<form action="saludar.php" method="post">

        Dime tu nombre: <input type="text" name="nombre" /> <br />
        <input type="submit" value="Saludar" /><br />
</form>

<?php
        }
        else
                echo "Hola " . $_POST["cuantos"];
?>
</body>
</html>
```

Y este sería un ejemplo más completo, que usara "PHP_SELF" para funcionar correctamente aunque cambiáramos el nombre del fichero que contiene el formulario:

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

	if ( ! isset($_POST["nombre"]) )
	{
?>
<form action="action="<?php echo $_SERVER["PHP_SELF"]; ?>"  method="post">

	Dime tu nombre: <input type="text" name="nombre" /> <br />
	<input type="submit" value="Saludar" /><br />
</form>

<?php
	}
	else
		echo "Hola " . $_POST["cuantos"];
?>
</body>
</html>
```

4.- Condiciones en PHP

En casi cualquier programa real, deberemos comprobar si se cumple una cierta condición. En PHP, la forma de hacerlo es empleando la orden if ("si", en inglés), seguido por la condición entre paréntesis, e indicando finalmente entre llaves los pasos que queremos dar si se cumple la condición, así :

```
if (condición)
	{
		sentencias
	}
```

Un primer ejemplo (condic01.php) podría ser:

```
<html>
<body>
<?php
  // condic01.php - Condiciones con "if", 1
  if (3>2) {
    echo "3 es mayor que 2";
  }
?>
</body>
</html>
```

Lo habitual es no comparar valores prefijados, sino el valor de una variable, que previamente hayamos leído de algún formulario o de una base de datos. Un segundo ejemplo (condic02.php) que compare un variable con un valor podría ser:

```
<html>
<body>
<?php
  // condic02.php - Condiciones con "if", 2
  $numero = 3;
  if ($numero>2) {
    echo "El numero es mayor que 2";
  }
?>
</body>
</html>
```

También podemos usar variables "booleanas", con valor de verdad **true** o **false** (también se pueden usar en mayúsculas).

Los operadores de **comparación** que podemos emplear son:

Operador	**Nombre**	**Ejemplo**	**Es cierto cuando:**
==	Igual	$a == $b	$a es igual a $b
===	Idéntico	$a === $b	$a es idéntico (en valor y tipo) a $b
!=, <>	Distinto	$a != $b	$a es distinto a $b
!==	No idéntico	$a !== $b	$a es distinto en valor o tipo a $b
<	Menor que	$a < $b	$a es menor que $b
>	Mayor que	$a > $b	$a es mayor que $b
<=	Menor o igual	$a <= $b	$a es menor o igual que $b
>=	Mayor o igual	$a >= $b	$a es mayor o igual que $b

Un ejemplo de uso de algunos de ellos sería:

```
<html>
<body>
<?php
// condic03.php - Condiciones con "if", 3
$n = 5;
if ($n == 5)
    echo "n vale 5 <br />";

$verdad = true;
if ($verdad)
    echo "la condición es verdad <br />";

if ($n = 4)
    echo "n vale 4<br />";

if ("7" == 7)
    echo '"7" es igual a 7<br />';

if ("6" !== 6)
    echo '"6" no es idéntico a 6<br />';
?>
</body>
</html>
```

que escribiría

```
n vale 5
la condición es verdad
n vale 4
"7" es igual a 7
"6" no es idéntico a 6
```

Cuidado con la tercera línea del resultado: si, por error, hacemos algo como "if ($n = 4)", con un único símbolo de igualdad, no estamos comparando, sino asignando un valor. Si esa asignación tuviera el valor cero, se tomaría como equivalente a "false"; si tiene cualquier otro valor (como en nuestro caso), se considera equivalente a "true".

Si queremos comprobar varias condiciones a la vez, las podemos enlazar con "y", "o", "no", que se pueden escribir de dos formas distintas:

Operador	**Nombre**	**Ejemplo**	**Devuelve cierto cuando:**
&&, and	Y	($a>2) && ($b<3) ($a>2) and ($b<3)	Cuando las dos condiciones son verdaderas.
||, or	O	($a>2) \|\| ($b<3) ($a>2) or ($b<3)	Cuando al menos una de las condiciones es verdadera.
!	No	! ($a>2)	Niega el valor de la expresión.

Además, con la orden **"else"** podemos indicar qué pasos queremos dar si no se cumple la condición, así:

```
if (condición)
{
   sentencias
}
else
{
   sentenciasAlternativas
}
```

Un ejemplo que use "else" y estos operadores (condic03.php) sería:

```
<html>
<body>
<?php
  // condic04.php - Condiciones con "if" y "else", 1
  $nombre = "Juan";
  $num = 3.5;
  if (($nombre != "Juan") or (!($num>4.0))) {
    echo "El nombre no es Juan o el numero es menor o igual que 4";
  }
  else {
    echo "Se llama Juan y el numero es mayor que 4";
  };
?>
</body>
</html>
```

También podemos **comparar cadenas de texto**, igual que hacemos con los números (y de forma mucho más sencilla que en lenguaje C), así (condic04.php):

```
<html>
$ <body>
<?php
  // condic05.php - Comparar valor de una cadena
  $nombre = "Juan";
  if ($nombre == "Juan") {
    echo "El nombre es Juan";
  }
?>
</body>
</html>
```

Si necesitamos comprobar varias condiciones, puede ser más cómodo usar la orden "switch", en vez de varios "if" seguidos.

Un ejemplo de su manejo (condic05.php) es:

```
<html>
<body>
<?php
 // condic06php - Comprobar múltiples valores con "switch"
 $numero = 4;
 echo "El número es: ";
 switch($numero) {
   case 1:
     echo "1";
     break;
   case 2:
   case 3:
     echo "2 o 3";
     break;
   case 4:
   case 5:
     echo "4 o 5";
     break;
   default:
     echo "No esta entre 1 y 5";
 }
?>
</body>
</html>
```

Es decir:

- Después de "switch" se indica entre paréntesis el nombre de la variable que queremos comprobar.
- Se abren llaves.
- Cada valor que queremos evaluar se indica entre la palabra "case" y un símbolo de dos puntos.
- A continuación se indican todos los pasos que queremos que se den si la variable tiene ese valor, terminados con "break".
- Si queremos que se haga lo mismo para varios valores de la variables, se indican varios "case" seguidos, terminados en un único "break", como se ve en el ejemplo.
- Para detallar lo que se debe hacer si el valor de la variable no es ninguno de los previstos, se usa la orden "default".

Nuevamente, al igual que ocurría con "if", también podremos manejar fácilmente cadenas con "switch" (cosa que no pasa en el lenguaje C, pero sí en otros más modernos como C#):

```
<html>
<body>
<?php
 // condic07.php - "switch" y cadenas de texto
 $nombre = "Paco";
 switch($nombre) {
   case "Juan":
     echo "El nombre es Juan";
     break;
   case "Paco":
     echo "El nombre es Paco";
     break;
```

```
    default:
        echo “El nombre no es Juan ni Paco
}
?>
</body>
</html>
```

5. Instrucciones repetitivas

Si queremos hacer que una sección de nuestro programa se repita mientras se cumpla una cierta condición, usaremos la orden **"while"**.

Si queremos comprobar la condición antes de comenzar, la sintaxis es

```
while  (condición)
  sentencia;
```

Es decir, la sentencia se repetirá mientras la condición se cierta. Si la condición es falsa ya desde un principio, la sentencia no se ejecuta nunca. Si queremos que se repita más de una sentencia, basta agruparlas entre { y }.

Un ejemplo muestre los números del 1 al 10 (repet1.php) sería

```
<html>
<body>
<?php
        // repet1.php, mostrar los números del 1 al 10 usando "while"
        $numero = 1;
        while ($numero <= 10) {
                echo $numero, "-";
                $numero ++;
                 }
?>
</body>
</html>
```

Que daría como resultado

```
1-2-3-4-5-6-7-8-9-10-
```

Lo que hace este programa es: comenzar con el valor 1 para la variable "numero". Mientras que ese número sea menor o igual que 10, lo escribe en pantalla, escribe un guión, y aumenta el valor de la variable "número", para pasar al siguiente.

La forma de sumar 1 al valor de una variable (muy frecuente cuando repetimos algo) es la que acabamos de ver:

```
                $numero ++;
```

Si queremos que aumente de 2 en 2, o de 3 en 3, la sintaxis cambia ligeramente:

```
                $numero += 3;
```

De igual modo, podemos disminuir en uno el valor de una variable con "--", o restarle un cierto número con "-=", o multiplicarlo por un número con "*=".

Existe otra variante de la orden "while", para cuando necesitemos que la condición se compruebe al final (útil por ejemplo para comprobar una clave de acceso, que debemos pedir al usuario al menos una vez, antes de verificar si es correcta). Se trata del conjunto **"do..while"**, como en este ejemplo (repet2.php):

```
<html>
<body>
<?php
 // repet2.php, mostrar los números del 0 al 10 usando "do-while"
 $numero = 0;
 do {
   echo $numero, "-";
   $numero ++;
 } while ($numero <= 10);
?>
</body>
</html>
```

Su resultado sería

```
0-1-2-3-4-5-6-7-8-9-10-
```

Finalmente, tenemos también la orden **"for"**, que típicamente se usa cuando sabemos cuantas veces queremos que se repita algo. Su formato es:

```
for   (valorInicial; CondiciónRepetición; Incremento)
   Sentencia;
```

Por ejemplo, para escribir los números pares del 10 al 20, podríamos usar una variable que tuviera como valor inicial 10, que se repita mientras tenga una valor menor o igual que 20, y que aumente 2 unidades en cada pasada, así (repet3.php):

```
<html>
<body>
<?php
        // repet3.php, mostrar los números pares del 10 al 20 usando "for"
        for ($numero=10; $numero<=20; $numero+=2) {
                echo $numero, "<br />";
         }
?>
</body>
</html>
```

6. Arrays (Cadenas o Vectores)

En los lenguajes "fuertemente tipados", como C, C++, php y C#, cada variable se declara con un tipo de datos asociado, y no se puede cambiar de tipo de datos.

En estos casos, un **"array"** es un conjunto de datos, todos ellos del mismo tipo, que se almacenan juntos y a los que se accede usando el mismo nombre de variable pero especificando la posición concreta en que está el dato individual que nos interesa.

En PHP, los tipos de datos asociados a las variables son mucho menos estrictos, porque se deducen del contexto y pueden cambiar. Por eso, los arrays también van a ser algo mucho más amplio que en un lenguaje de programación convencional.

Vamos a comenzar con un primer ejemplo que manipule **números prefijados** en un array:

```
<html>
<body>
<?php
// arrays01.php - array con datos prefijados

$datos = array(20, 41, 36, 12.8, 26);
echo "Mostrando los datos con for: ";
for ($i=0; $i<5; $i++)
    echo $datos[$i] , " - ";
echo "<br />";
echo "Mostrando con print\_r: ";
print_r($datos);
echo "<br />";
?>
</body>
</html>
```

Este ejemplo mostraría:

```
Mostrando los datos con for: 20 - 41 - 36 - 12.8 - 26 -
Mostrando con print_r: Array ( [0] => 20 [1] => 41 [2] => 36 [3] => 12.8 [4] => 26 )
```

"print_r" muestra todo el contenido del array en un formato legible. La alternativa "tradicional" es recorrer todo el array usando "for" o bien otra construcción que veremos un poco más adelante.

Pero un array en PHP puede contener **datos de distintos tipos**:

```
<html>
<body>
<?php
// arrays02.php - array con datos prefijados de distinto tipo

$datos = array(20, 41, "Ejemplo", 12.8, "Otro ejemplo");
echo "Mostrando los datos con for: ";
for ($i=0; $i<5; $i++)
    echo $datos[$i] , " - ";
echo "<br />";
echo "Mostrando con print_r: ";
print_r($datos);
echo "<br />";
?>
</body>
</html>
```

que haría:

```
Mostrando los datos con for: 20 - 41 - Ejemplo - 12.8 - Otro ejemplo -
Mostrando con print_r: Array ( [0] => 20 [1] => 41 [2] => Ejemplo [3] => 12.8 [4] => Otro ejemplo )
```

Podemos incluso hacer que los índices no sean consecutivos, sino que vayan **en cualquier orden**, pero en este caso ya no podremos usar "for" para recorrerlos. En su lugar, usaremos "foreach", que permite extraer cada uno de los datos, dándole un nuevo nombre:

```
<?php
// arrays03.php - índices no consecutivos

$datos = array(5=>20, 0=>41, 7=>"Ejemplo", 3=>12.8, "Otro ejemplo");
echo "El dato 0 es $datos[0]<br />";
echo "Mostrando con print\_r: ";
print_r($datos);
echo "<br />Mostrando los datos con foreach: ";
foreach ($datos as $unDato)
    echo $unDato , " - ";
echo "<br />";
?>
```

que tendría como resultado:

El dato 0 es 41
Mostrando con print_r: Array ([5] => 20 [0] => 41 [7] => Ejemplo [3] => 12.8 [8] => Otro ejemplo)
Mostrando los datos con foreach: 20 - 41 - Ejemplo - 12.8 - Otro ejemplo -

También podemos usar **textos como índice**, añadir valores en cualquier momento, borrarlos con "unset" y comprobar si existen con "isset". Para saber el tamaño (cantidad de datos), usaremos "count". Para añadir al final, dejaremos los corchetes vacíos. Y podemos usar "foreach" para extraer tanto el índice como el valor, así:

```
<?php
// arrays04.php - texto como índices, añadir, borrar...
$datos = array(5=>20, "hola"=>41, 7=>"Ejemplo", 3=>12.8, "Otro
ejemplo");
echo "Datos iniciales: ";
print_r($datos);
echo "<br /><br />Añadimos el dato 2 y el dato \"yo\": ";
$datos[2] = 23;
$datos["yo"] = 32;
print_r($datos);
echo "<br /><br />Añadimos un dato al final: ";
$datos[] = 45;
print_r($datos);
echo "<br /><br />Borramos un dato: ";
unset($datos["hola"]);
if (! isset($datos["hola"]))
    echo "\"hola\" se ha borrado ";
print_r($datos);
echo "<br /><br />Mostrando los datos con foreach: ";
foreach ($datos as $clave => $valor)
    echo $clave, "=", $valor , " - ";
echo "<br /><br />Cantidad de datos: ", count($datos);
echo "<br />";
?>
```

que nos respondería:

```
Datos iniciales: Array ( [5] => 20 [hola] => 41 [7] => Ejemplo [3] => 12.8 [8] => Otro ejemplo )

Añadimos el dato 2 y el dato "yo": Array ( [5] => 20 [hola] => 41 [7] => Ejemplo [3] => 12.8 [8] => Otro ejemplo [2]
=> 23 [yo] => 32 )

Añadimos un dato al final: Array ( [5] => 20 [hola] => 41 [7] => Ejemplo [3] => 12.8 [8] => Otro ejemplo [2] => 23
[yo] => 32 [9] => 45 )
```

```
Borramos un dato: "hola" se ha borrado Array ( [5] => 20 [7] => Ejemplo [3] => 12.8 [8] => Otro ejemplo [2] => 23 [yo] => 32 [9] => 45 )

Mostrando los datos con foreach: 5=20 - 7=Ejemplo - 3=12.8 - 8=Otro ejemplo - 2=23 - yo=32 - 9=45 -

Cantidad de datos: 7
```

Y hemos manejado otros arrays existentes en PHP anteriormente: $_GET para los datos de formularios que se pasen de forma visible, $_POST para los datos de formularios pasados de forma invisible y $_SERVER para saber cierta información sobre el servidor, como por ejemplo:

- $_SERVER["PHP_SELF"] es el nombre de la página actual (incluyendo la ruta pero no el servidor).
- $_SERVER["REMOTE_ADDR"] es la dirección IP desde la que se ha consultado la página.
- $_SERVER["HTTP_USER_AGENT"] es el "agente" desde el que se ha pedido la página. Típicamente será un navegador web, y este texto contendrá detalles como el nombre del navegador, la versión y el sistema operativo anfitrión.

7. Algunas funciones útiles ya incluidas en PHP

7.1. Manipulación de cadenas de texto

Vamos a ver un ejemplo que incluya algunas de las funciones de uso más habitual, y luego comentaremos algo sobre cada tipo de función:

```
<html>
<body>
<?php
	$frase="EjEMplo de FRASE";
	echo "Texto original: " . $frase . "<br />";
	echo "Mayusculas: " . strtoupper($frase) . "<br />";
	echo "Minusculas: " . strtolower($frase) . "<br />";
	echo "Palabras en mayusculas: " . ucwords($frase) . "<br />";
	echo "Palabras en mayusculas y resto en minúsculas: " .
	ucwords(strtolower($frase)) . "<br />";
	// La siguiente, solo en PHP 5.30 o superior
	// echo "Primera en minusculas: " . lcfirst($frase) . "<br />";
	echo "Palabras sin espacios: " . str_replace(" ", "", $frase) . "<br />";
	echo "Longitud: " . strlen($frase) . "<br />";
	echo "Primera letra: " . $frase[0] . "<br />";
	echo "\"de\" esta en la posicion: " . strpos($frase, "de") . "<br />";

	echo "Palabras individuales:<br />";
	$palabras = explode(" ", $frase);
	foreach ($palabras as $unaPalabra)
		echo "- " . $unaPalabra . "<br />";
	$nuevaFrase = implode("-", $palabras);
	echo "Vuelto a juntar: " . $nuevaFrase . "<br />";

	echo "Al reves: " . strrev($frase) . "<br />";
	$caracteresEspeciales = "<hola> &tal";
	echo "Una frase con caracteres especiales: ". htmlspecialchars($caracteresEspeciales) . "<br />";
```

```
        $fraseLarga = "Ejemplo de una frase mas larga, que vamos a partir en varias lineas";
        echo "Frase partida: <br />"  . wordwrap($fraseLarga, 15, "<br />");

        $conEspacios = " de ";
        echo "<br />";
        echo "Sin espacios iniciales: -" . ltrim($conEspacios) . "-<br />";
        echo "Sin espacios finales: -" . rtrim($conEspacios) . "-<br />";
        echo "Sin espacios iniciales ni finales: -" . trim($conEspacios) . "-<br />";
?>
</body>
</html>
```

Y se mostraría en pantalla algo como:

```
Texto original: EjEMplo de FRASE
Mayusculas: EJEMPLO DE FRASE
Minusculas: ejemplo de frase
Palabras en mayusculas: EjEMplo De FRASE
Palabras en mayusculas y resto en minúsculas: Ejemplo De Frase
Primera en minusculas: ejEMplo de FRASE
Palabras sin espacios: EjEMplodeFRASE
Longitud: 16
Primera letra: E
"de" esta en la posicion: 8
Palabras individuales:
- EjEMplo
- de
- FRASE
Vuelto a juntar: EjEMplo-de-FRASE
Al reves: ESARF ed olpMEjE
Una frase con caracteres especiales: <hola> &tal
Frase partida:
Ejemplo de una
frase mas
larga, que
vamos a partir
en varias
lineas
Sin espacios iniciales: -de -
Sin espacios finales: - de-
Sin espacios iniciales ni finales: -de-
```

Es decir...

- Podemos convertir a mayúsculas con "strtoupper" o a minúsculas con "strtolower", convertir a mayúsculas sólo el principio de cada palabra con "ucwords"
- Se puede reemplazar una letra o una subcadena por otra, todas las veces que aparezca, usando "str_replace".
- Para saber la longitud de una cadena, emplearemos "strlen", y corchetes si queremos extraer una letra concreta, o "strpos" para saber si contiene una cierta subcadena.

- Podemos "partir" una cadena en trozos, usando un cierto separador, con "explode". Obtendremos que podríamos recorrer con "for" o "foreach". También podemos dar el paso contrario: crear una cadena a partir de los elementos de un array, con "implode".
- Si el texto contiene caracteres de control, como "", podemos dejarlos tal cual para que se interpreten como un cambio a negrita en una página web, o bien usar "htmlspecialchars" para que se conviertan a texto visible.
- Hay otras muchas manipulaciones de uso poco frecuente, como "strrev" para dar la vuelta a una cadena o "wordwrap" para formatear un párrafo a lo largo de varias líneas, con una anchura máxima.
- "ltrim" elimina los espacios a la izquierda de una cadena (al principio), "rtrim" los de la derecha (el final) y "trim" los de ambos lados.

7.2. Funciones de fecha y hora

Nuevamente, comencemos por un ejemplo que muestre las posibilidades más habituales:

```
<html>
<body>
<?php
        $fecha = date("d-m-y"); /\* Dia, mes y año con dos cifras \*/
        echo "La fecha de hoy es: " . $fecha . "<br />";
        $mes[1] = "enero"; $mes[2] = "febrero"; $mes[3] = "marzo";
        $mes[4] = "abril"; $mes[5] = "mayo"; $mes[6] = "junio";
        $mes[7] = "julio"; $mes[8] = "agosto"; $mes[9] = "septiembre";
        $mes[10] = "octubre"; $mes[11] = "noviembre"; $mes[12] = "diciembre";
        echo date("j") /\* Dia sin 0 inicial \*/
        . " de ".  $mes[date("n")]  /\* Mes sin 0 inicial \*/
         . " de ".  date("Y") /\* Año con 4 cifras \*/
         . "<br />";
        echo "Día de la semana (0 a 6): " . date("w"). "<br />";
        $dia[1] = "lunes"; $dia[2] = "martes"; $dia[3] = "miércoles";
        $dia[4] = "jueves"; $dia[5] = "viernes"; $dia[6] = "sábado";
        $dia[0] = "domingo";
        echo "El día de la semana es: " . $dia[date("w")] . "<br />";
        echo "o bien: " . date("l") . "<br />";
        echo "Día del año (0 a 365): " . date("z"). "<br />";
        echo "Hora actual: "
         . date("H:i:s"). "<br />"; /\* Hora de 00 a 23, mins y segs de 00 a 59 \*/
?>
</body>
</html>
```

Que mostraría algo como

```
La fecha de hoy es: 09-12-13
9 de diciembre de 2013
Día de la semana (0 a 6): 1
El día de la semana es: lunes
o bien: Monday
```

```
Día del año (0 a 365): 342
Hora actual: 22:29:36
```

Donde:

- La fecha: en general se sabe a partir de la función "date". Como parámetro a esta función se le indica entre comillas la información que queremos saber, que puede ser:
- "d" es el número de día, con dos cifras ("j" sería usando sólo una cifra para los días 1-9).
- "m" es el número de mes, don dos cifras (o "n", con una cifra cuando sea posible).
- "y" es el número de año, con dos cifras("Y" sería con 4 cifras).
- "H" es la hora, en formato de 24 horas ("h" sería en formato de 12 horas).
- "i" es el número de minutos, con ceros iniciales.
- "s" son los segundos, con ceros iniciales.
- "w" es el número del día de la semana (empezando en 0, para el domingo).
- Para el nombre del día, podríamos usar "l", pero lo obtendremos en inglés, de modo que en general será preferible tener guardados los nombres de los días en español en un array. La misma idea se podría aplicar a los meses.

7.3. Números al azar

Nuevamente, comencemos por un ejemplo que muestre las posibilidades más habituales:

```
<html>
<body>
<?php
$numAzar1 = rand();          // Cualquier valor (quizá sólo hasta 32767)
$numAzar2 = rand() % 9;      // Del 0 al 8
$numAzar3 = rand() % 11 + 5; // Del 5 al 15
$numAzar4 = rand(10,20);     // Del 10 al 20
$numAzar5 = mt_rand();       // Cualquier valor (quizá mucho mayor que 32767)
$numAzar6 = mt_rand(20,30);          // Del 20 al 30
$numAzar7 = mt_rand(1,999) / 1000;  // Del 0,001 al 0,999
echo $numAzar1, " - ", $numAzar2, " - ", $numAzar3, " - ",
   $numAzar4, " - ", $numAzar5, " - ", $numAzar6, " - ", $numAzar7;
?>
</body>
</html>
```

Y se mostraría en pantalla algo como:

```
7443 - 3 - 10 - 13 - 2052892426 - 24 - 0.995
```

Y los detalles son:

- rand() devuelve un valor entre 0 y "getrandmax()", un valor que en algunas plataformas como Windows solo alcanza hasta 32767.
- La forma "clásica" de obtener un número entre dos valores es hacer el módulo para limitar el rango y luego sumar el valor mínimo
- PHP permite otra forma más compacta: indicar el valor mínimo y el máximo
- mt_rand() es una variante más moderna, que puede generar números mucho más grandes y aun así es muy rápida
- mt_rand() también permite indicar un valor mínimo y uno máximo

- Si queremos obtener un número del 0 al 1, podemos generar un número entero y dividirlo entre su valor máximo
- En versiones antiguas de PHP (antes de la 4.2.0), podría ser necesario indicar una semilla antes de empezar a generar números al azar. Lo razonable es que esa semilla fuera el reloj interno del ordenador, lo que se conseguiría con "srand(time());"

8. Creando funciones en PHP

Hemos visto cómo usar algunas de las funciones existentes en PHP, y que nos permiten hacer cosas como convertir un texto a mayúsculas, generar un número al azar u obtener la fecha/hora actual.

Vamos a ver cómo crear nuestras propias funciones, algo que puede tener varios motivos interesantes:

- Evitar que se repitan trozos de programa: podremos agrupar varias órdenes en una sola.
- Por otra parte, esto debería ayudar a que nuestro programa sea más legible. Algo como "obtenerArticulo" puede resultar mucho más legible que una serie de 10 órdenes escritas en inglés y que manejan varias variables...
- Además, esto permitiría repartir el trabajo entre varias personas con mucha más facilidad, e incluso reutilizar el trabajo más adelante, aunque veremos formas más eficientes.

En PHP, el formato general para definir una función será este, si no devuelven resultado:

```
function nombre_de_funcion ()   {
   orden1;
   orden2;
   orden3;
   ...

}
```

o en el caso de funciones que devuelven un resultado bien numérico, texto o cualquier otro:

```
function nombre_de_funcion (parametro1, parametro2...)   {
   orden1;
   orden2;
   orden3;
   ...
   return resultado;
}
```

(Hay que recordar que "parametro1", "parametro2" y "resultado" normalmente serán variables, y en ese caso deberían empezar con el símbolo $)

Como primer ejemplo, podríamos crear una función que escriba un título a partir de un cierto texto, convirtiéndolo a mayúsculas salvo el comienzo de cada palabra, así:

```
// function01.php - Primer ejemplo de función
function escribirTitulo($texto)   {
   echo "<h1>". ucwords(strtolower($texto)) . "</h1>";
}

escribirTitulo("hola");
```

Una función matemática que devolviera el resultado de elevar un número al cubo podría ser:

```
// function02.php - Función que devuelve un valor
function cubo($numero)   {
    $resultado = $numero \* $numero \* $numero;
    return $resultado;
}

echo cubo(4);
```

Si queremos modificar el valor de un parámetro, debemos preceder su nombre con el símbolo "&" (es lo que se conoce como "pasar el parámetro por referencia"; lo opuesto, que hemos hecho en los dos ejemplos anteriores, es "pasar el parámetro por valor"), así:

```
<html>
<body>
Bienvenido!<br>
<?php
        // function03.php - Modificando un parámetro
        function incrementa (&$variable) {
        $variable++;
        }

        $a=5;
        echo "a vale ", $a, "<br />";
        incrementa($a);
        echo "Ahora a vale ", $a, "<br />";   // Ahora $a vale 6
        ?>
</body>
</html>
```

(Esto puede ser casi imprescindible en ciertas circunstancias, como por ejemplo cuando una función deba devolver más de un valor).

9. Inclusión de ficheros

Es muy frecuente que las páginas web de un cierto sitio tengan una cabecera común y un pie común, que son iguales (o muy parecidos) en todas ellas.

Eso supone que si tenemos 20 páginas parecidas y queremos hacer un cambio en todas ellas, deberíamos ir modificándola una por una.

Esto es un caso bastante más frecuente de lo que parece. Ocurre por ejemplo cuando tenemos toda una estructura de un sitio web, y todas las páginas tienen un menú en la parte superior. Si queremos hacer un cambio en el menú de todas las páginas, la alternativa "convencional" sería modificarlas una a una (o quizá usar alguna herramienta que reemplace textos de forma repetitiva).

Una alternativa cómoda es que esa cabecera, o ese pie, o un menú, esté en un fichero independiente, y lo "incluyamos" desde cada una de las páginas. Así, si queremos hacer cambios, sólo será necesario retocar el fichero de la cabecera, o del pie, o del menú, y todas las demás páginas lo reflejen inmediatamente. La forma de conseguirlo sería la siguiente:

El fichero que se enlazaría desde todos podría ser un fragmento de fichero HTML totalmente normal, sin cabeceras como HEAD y BODY:

```
<a href="menu1.php">Opción 1</a>
<a href="menu2.php">Opción 2</a>
```

Y los ficheros que accedieran a él, usarían la orden "include", a la que se le indica el nombre de fichero que se quiere insertar en ese punto (por ejemplo, menu.php):

```
<html>
<body>
Principio de la pagina<br>
Menu: <?php include "menu.php" ; ?><br />
Fin de la pagina<br>
</body>
</html>
```

La persona que visitara nuestra web vería una única página totalmente normal, el resultado de fusionar nuestra página "principal" con todas las páginas "auxiliares" que hayamos incluido:

```
<html>
<body>
Principio de la pagina<br>
Menu: <a href="menu1.php">Opción 1</a>
<a href="menu2.php">Opción 2</a><br />
Fin de la pagina<br>
</body>
</html>
```

Pero cuando estamos trabajando con PHP, eso nos da más juego: podemos crear una "biblioteca" que recopile nuestras funciones habituales, y enlazarlas desde cualquier otra página para tener acceso a dichas funciones. Así, partiendo del ejemplo de la lección anterior, podríamos crear un fichero llamado "biblioteca.php" que contuviera

```
<?php
function escribirTitulo($texto)   {
    echo "<h1>". ucwords(strtolower($texto)) . "</h1>";
}
?>
```

y otro fichero llamado "ejemplo.php" que la usara, podría ser así:

```
<html>
<body>
<?php
include "biblioteca.php";
escribirTitulo("Bienvenido");
echo "Estamos utilizando la biblioteca";
?>
</body>
</html>
```

Nota: nuestros ficheros incluidos podrían tener un nombre terminado en ".inc" , ".html" o cualquier otro, ya que van a ser incluidos en uno que ya de por sí es ".php". De todas formas, es recomendable que su extensión sea ".php", para que los usuarios de nuestras páginas no tengan acceso al código fuente, aunque pudieran llegar a acertar el nombre del fichero (si el servidor sabe que es un fichero PHP, no dejará que el navegador vea su código fuente, sino que envía los resultados correspondientes, pero si no sabe que se trata de PHP -porque la extensión sea distinta-, quizá sí deje ver el código fuente).

10. Acceso a bases de datos desde PHP

10.1. Creación de una base de datos con PhpMyAdmin

Para poder acceder a una base de datos desde PHP usando SQL necesitamos antes tener una base de datos.

Para crearla, emplearemos PhpMyAdmin, una herramienta de administración para PHP y MySQL que se incluye como parte de la mayoría de paquetes "WAMP", como Uniform Server, XAMPP o EasyPHP.

Vamos a planificar antes de comenzar:

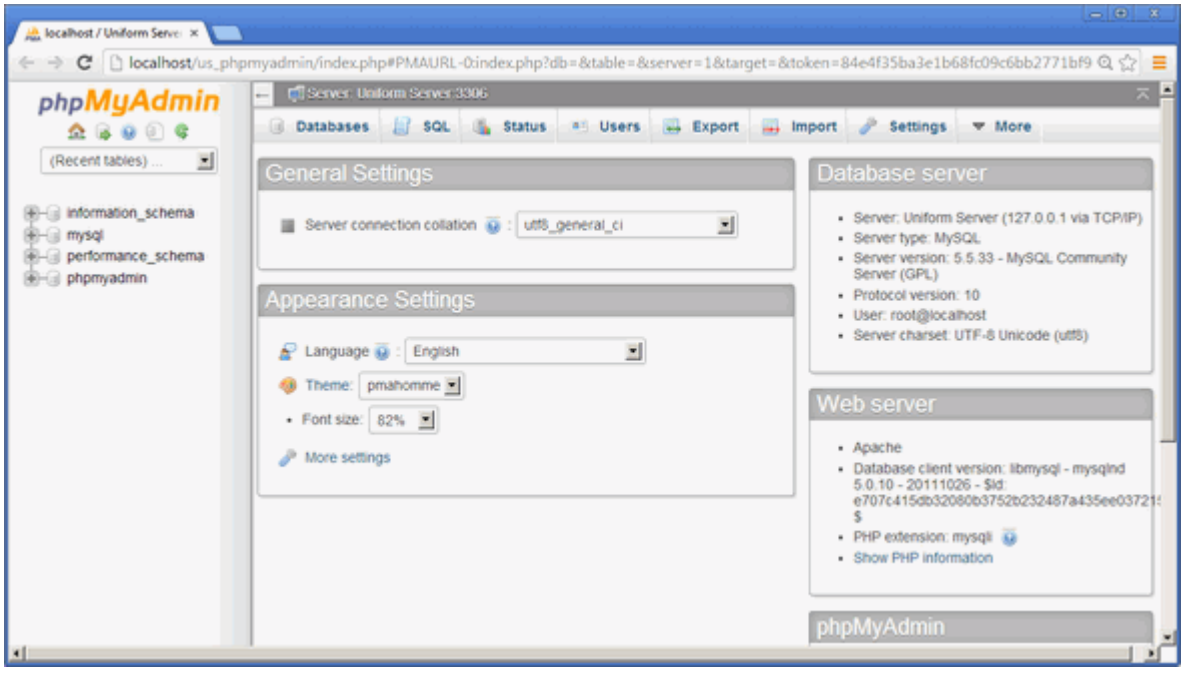
- Crearemos una base de datos que se llamará "prueba".
- Esta base de datos contendrá un único bloque de información, una tabla llamada "amigos".
- Para cada uno de esos amigos, guardaremos 4 datos (campos): nombre (que será texto de hasta 40 letras), email (texto hasta 50 letras), telefono (texto hasta 20 letras), fechanac (fecha de nacimiento, tipo de datos "fecha").

Los pasos que debemos seguir son:

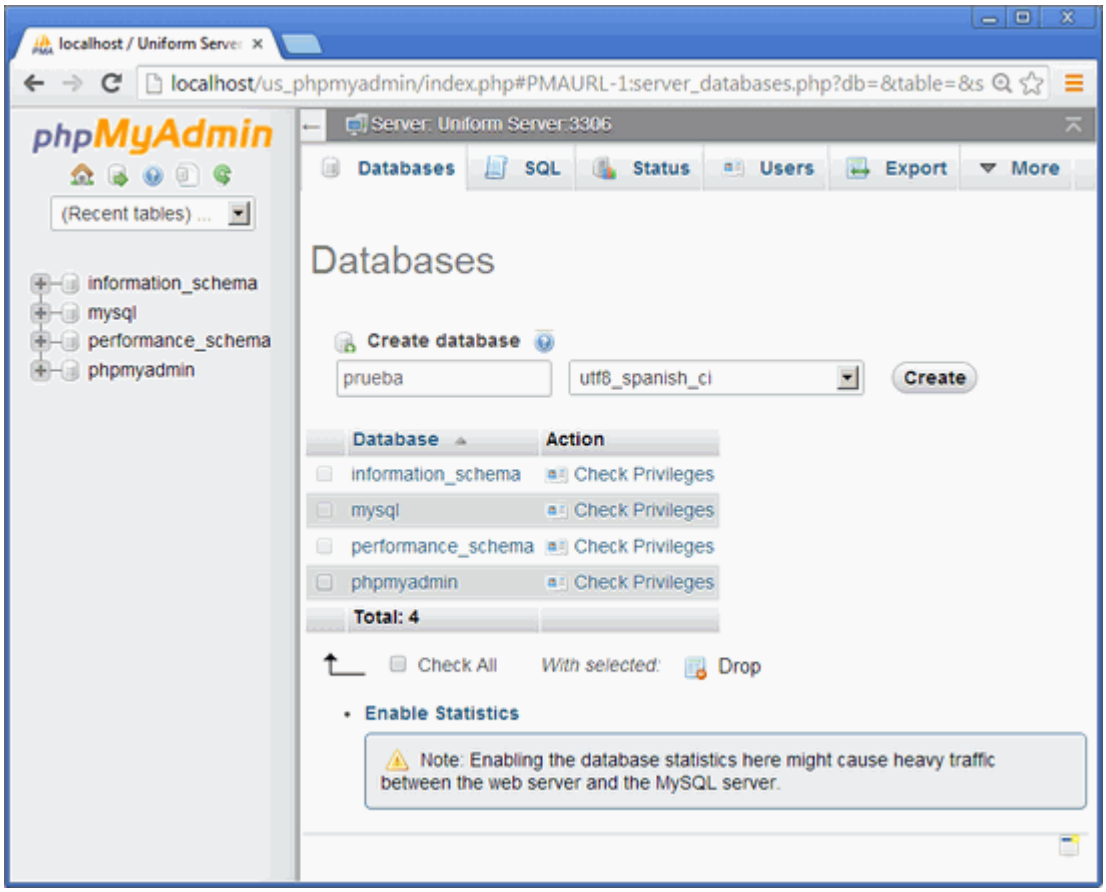
1.- Lanzar nuestro servidor web.

2.- Abrir PhpMyAdmin. En ocasiones tendremos que teclear la ruta "http://localhost/phpmyadmin" o alguna similar en nuestro navegador; en el caso de The Uniform Server, el panel de control nos muestra un botón llamado "phpMyAdmin", que nos permite abrirlo de forma sencilla:

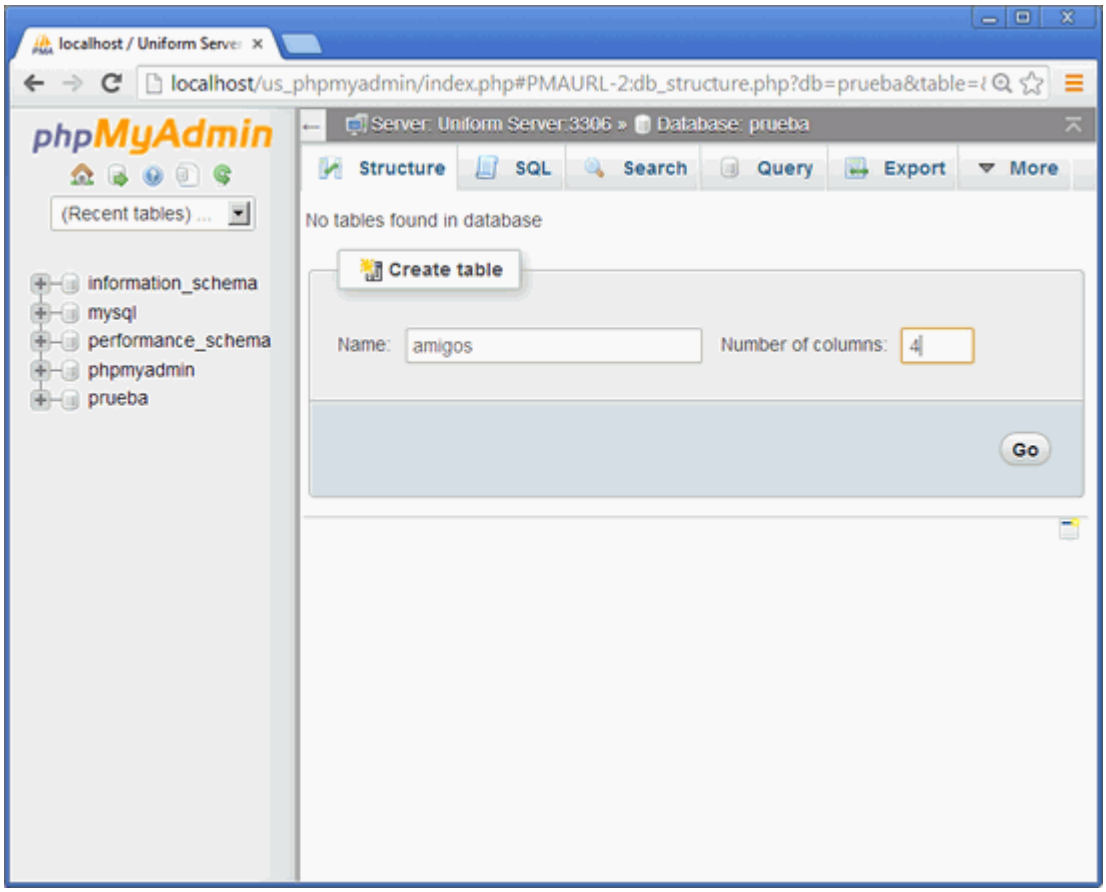
3.- Debería aparecer la ventana principal de PhpMyAdmin:



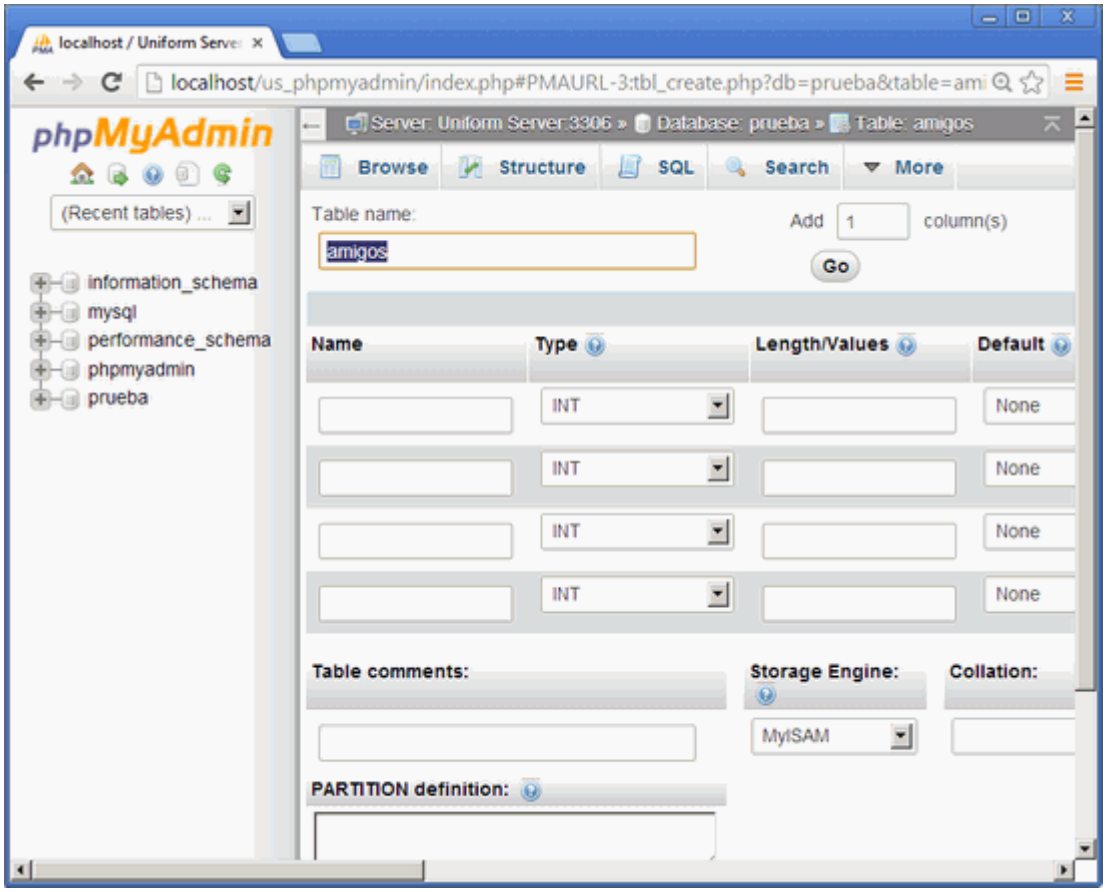
4.- En la pestaña "Databases", tendremos la opción para crear una nueva base de datos (Create database). En el recuadro introduciremos la palabra "prueba" como nombre para la base de datos y en el desplegable "Collation", escogeremos "utf8_spanish_ci" como juego de caracteres. Terminaremos pulsando el botón "Create".



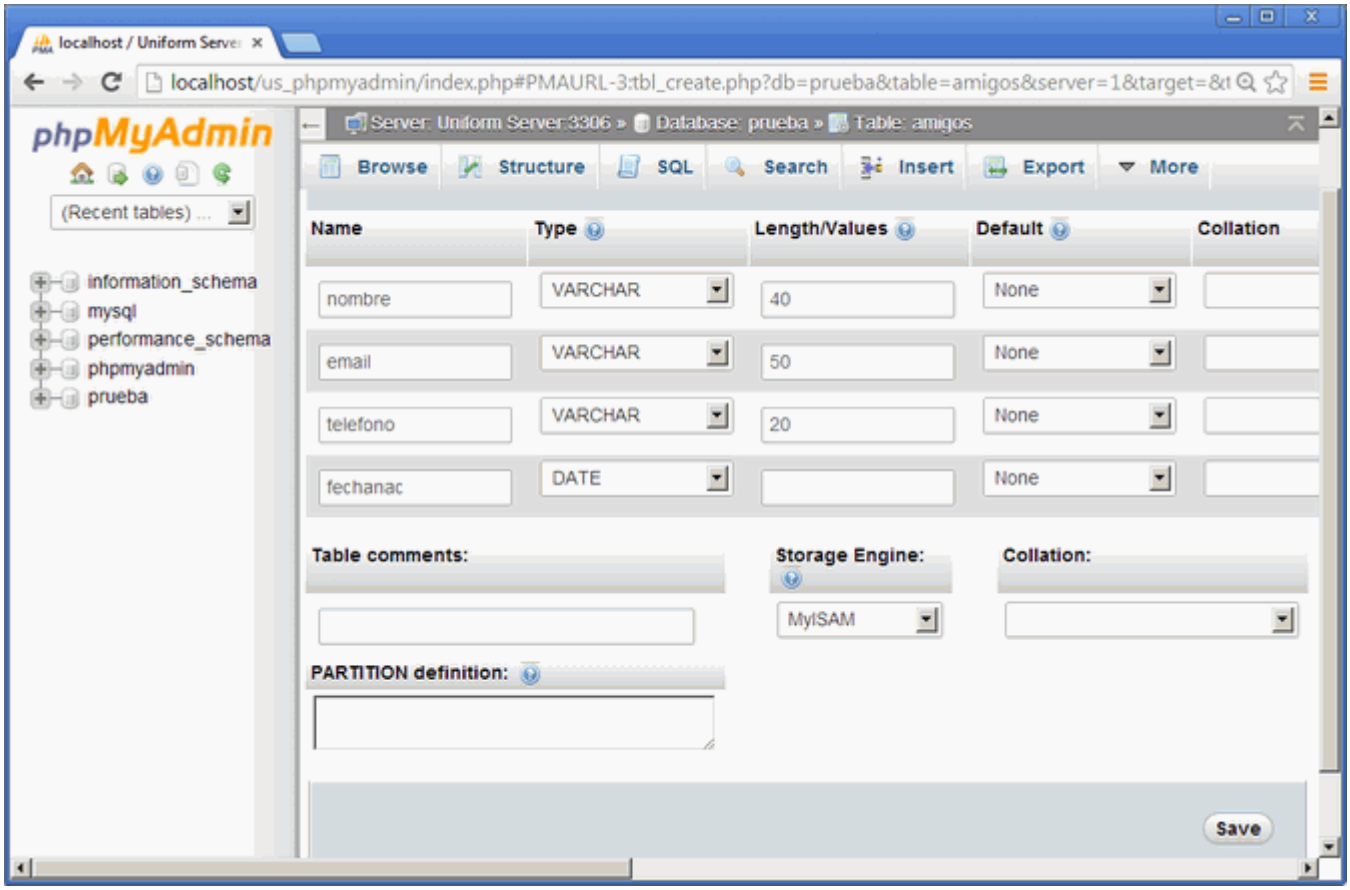
5.- Al hacer clic en "prueba" nos aparecerá el contenido de la base de datos, que todavía está vacía y un apartado "Create table" ("crear tabla") que espera que le digamos un nombre (name) y una cantidad de campos (number of columns). Como nombre, introduciremos la palabra "amigos", y como número de campos indicamos "4". Hacemos clic en el botón "Go" ("adelante"), para crear la tabla.



6.- Aparece la lista de introducción de datos para los campos:

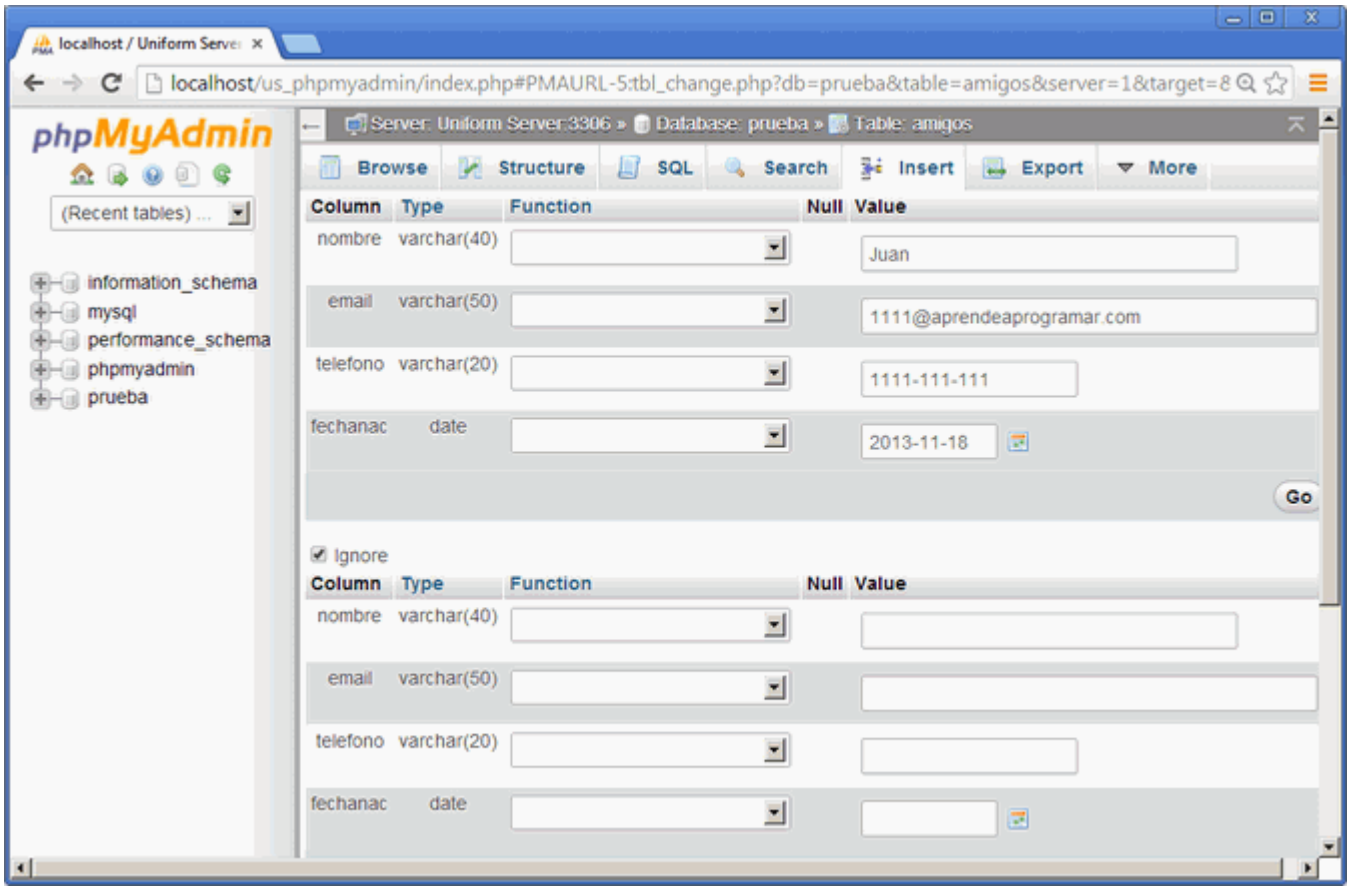


7.- Introducimos los valores que habíamos diseñado, utilizando el tipo de datos "VARCHAR" para los campos que van a guardar texto, indicando su longitud máxima, y empleando el tipo "DATE" para la fecha. Finalmente pulsamos el botón "Save" (grabar):



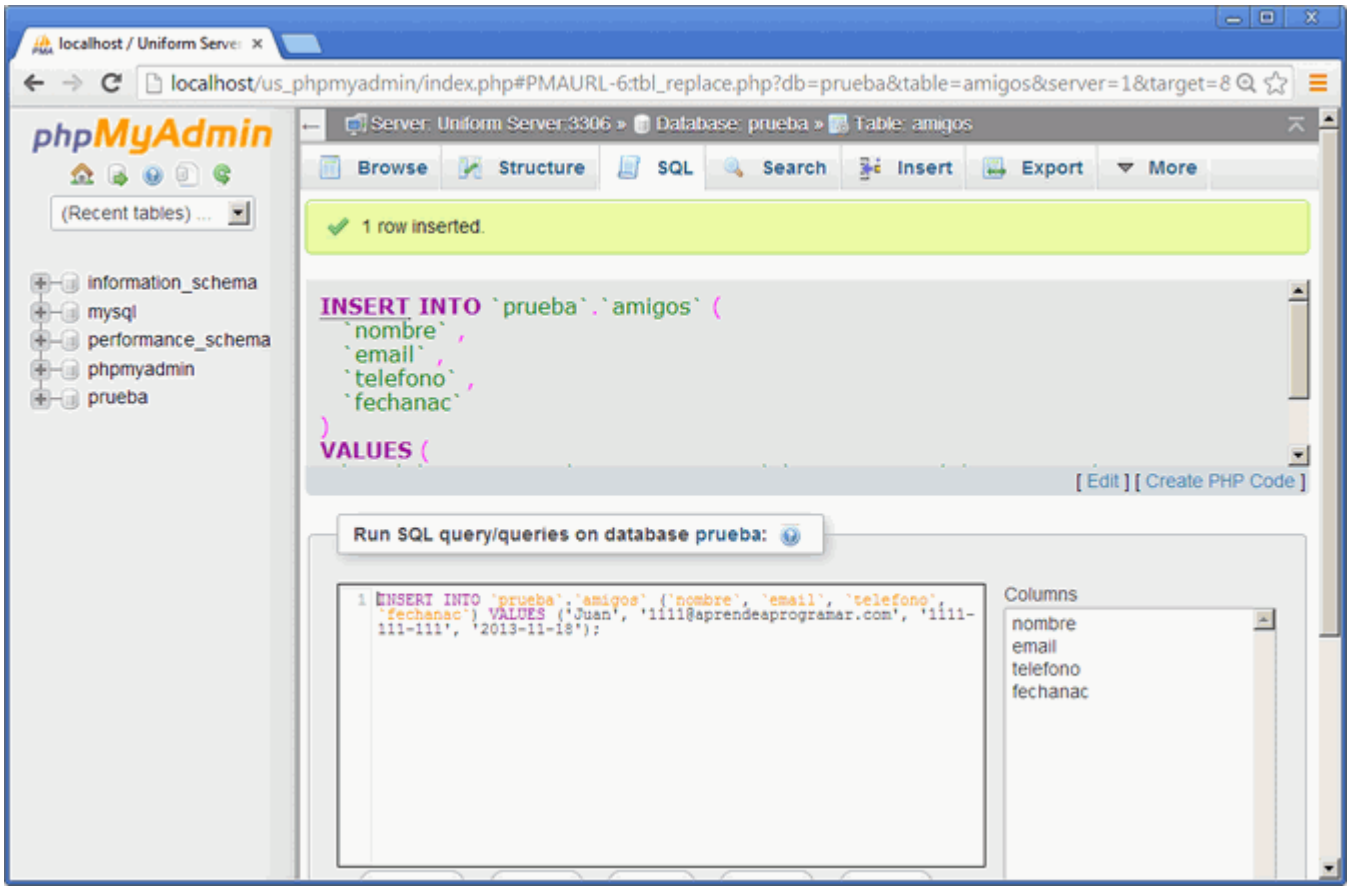
(Hay cosas que se podrían mejorar; por ejemplo, no hemos definido ninguna "clave", por lo que podríamos tener datos duplicados, y hemos dejado marcada la casilla "not null", por lo que necesariamente deberemos indicar una valor para cada uno de los campos, pero no se trata aquí de aprender a diseñar bases de datos, sino de jugar con PHP)

8.- Ya que estamos, podemos incluir un par de fichas (registros) desde PhpMyAdmin, para tener datos con los que jugar después desde nuestros mini-programas en PHP. Hacemos clic en la pestaña superior "Insert" (insertar):

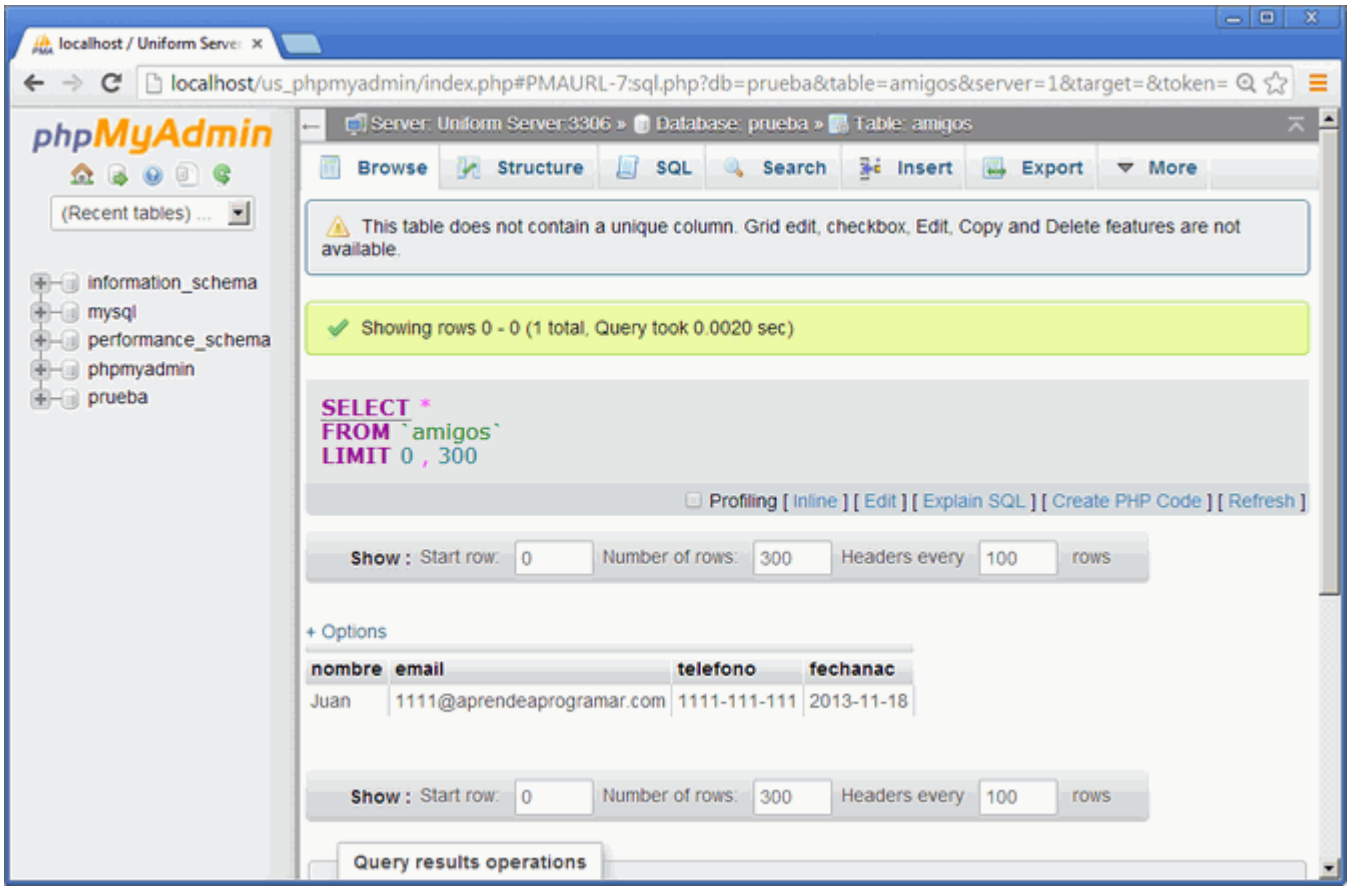


(Como se ve en el ejemplo, las fechas se deben introducir en formato AAAA-MM-DD).

El resultado de esta operación, nos dará una pista de qué deberemos hacer más adelante para introducir datos desde PHP:

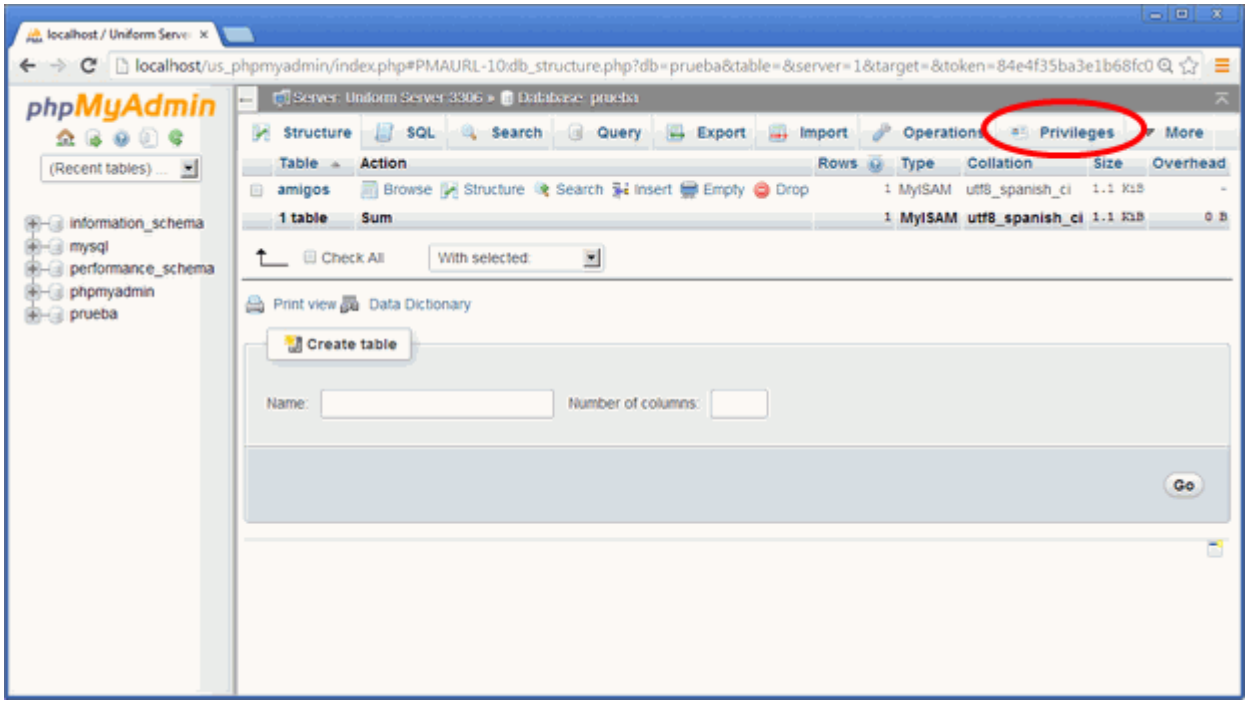


9.- Podemos comprobar que se han guardado los datos correctamente si usamos la pestaña "Browse" (echar un vistazo). El resultado nos debería confirmar que hemos introducido correctamente los datos. Se nos propone la consulta "SELECT * FROM `amigos` LIMIT 0,300", que es la que nos interesa para ver todos los datos que contiene nuestra tabla de amigos (nuevamente, consultas parecidas a esa serán las que usaremos desde PHP):

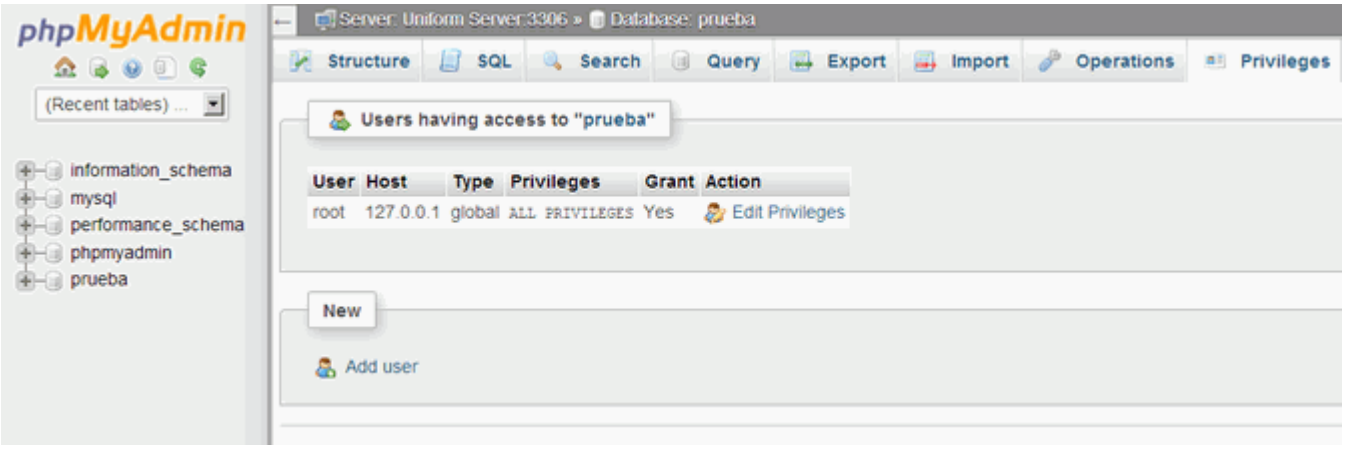


10.- Finalmente, queda una cosa por hacer para poder acceder a nuestros datos desde PHP. Debemos crear un usuario, que tenga permiso para acceder a esta base de datos.

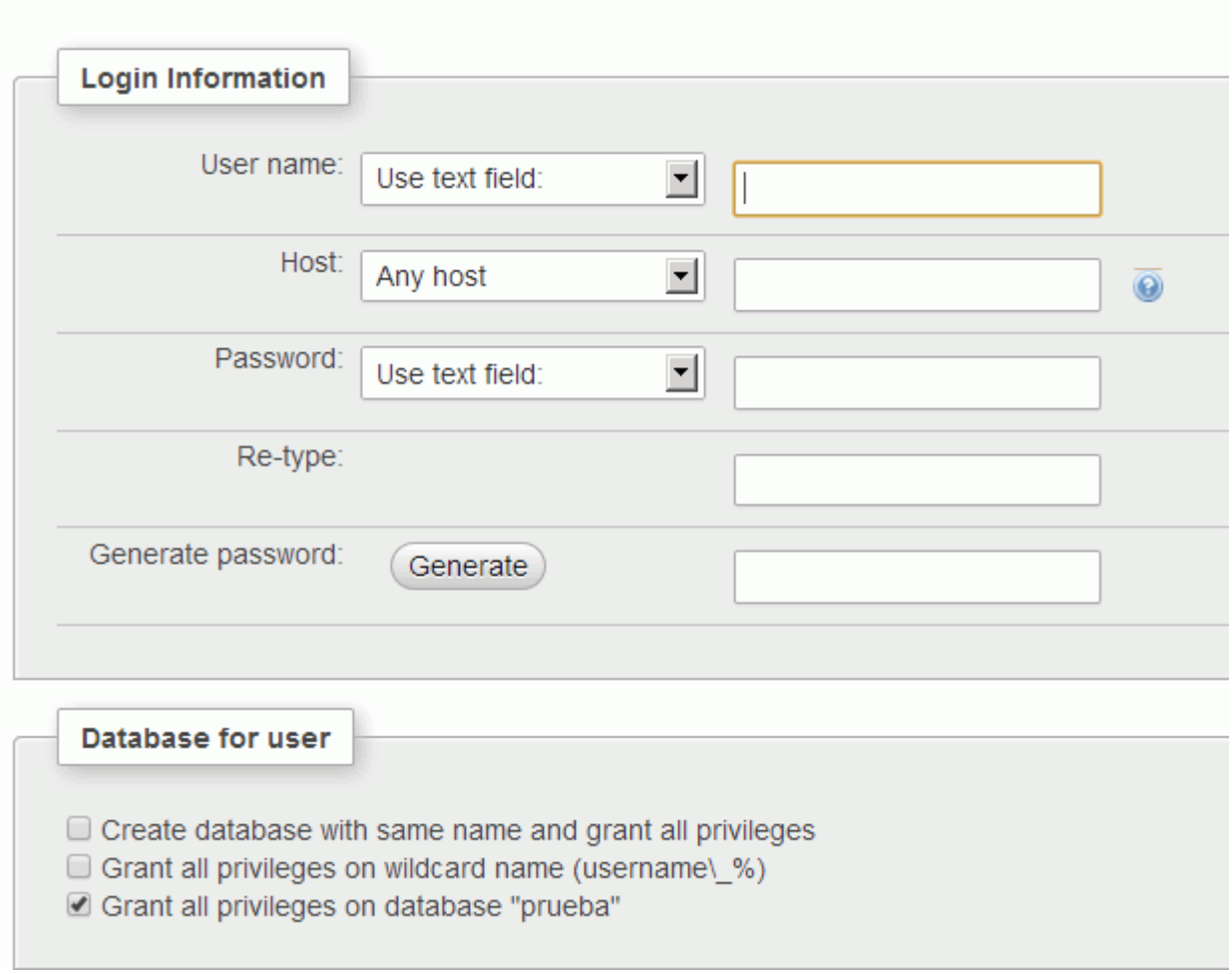
Para ello, volvemos a la base de datos, haciendo clic en la palabra "Prueba", y después pulsamos en la pestaña "Privileges" (privilegios):



A continuación, hacemos clic en "Add user" (agregar usuario):



Emplearé como nombre "usuario", y "1234" como contraseña (obviamente, en un proyecto real no deberían ser nombres y contraseñas tan triviales). Se pueden restringir los permisos para que sólo pueda realizar ciertas operaciones, pero no es nuestro caso. Como queremos poder practicar las operaciones más habituales desde PHP, dejaremos marcada la opción "Grant all privileges on database 'prueba'" (conceder todos los permisos en la base de datos "prueba"):



Listo. Aparentemente, ya podemos empezar a manipular datos desde PHP...

10.2. Consulta de bases de datos desde PHP

Para consultar desde PHP la información existente en una base de datos, los pasos serán básicamente los siguientes:

- Abrir la conexión con el servidor de bases de datos (como estamos en nuestro propio ordenador, será "localhost"), autentificándonos con nuestro nombre de usuario ("usuario") y nuestra contraseña ("1234"). Usaremos la orden "mysql_connect".
- Escoger la base de datos que nos interese (en nuestro caso, "prueba"), con la orden "mysql_select_db".
- Realizar la consulta en SQL que queramos, con "mysql_query". En nuestro ejemplo, querremos todos los datos de la tabla "amigos", para lo que serviría hacer "SELECT * FROM amigos". (Cuidado: en la órdenes de SQL, como SELECT, no se suele distinguir entre mayúsculas y minúsculas, pero en los nombres de las tablas es posible que sí).
- Para obtener cada uno de los registros, utilizaremos "mysql_fetch_array", que devolverá "falso" cuando no haya más datos, por lo que es frecuente recorrer los registros usando un "while". Para acceder a cada uno de los campos, se usa una sintaxis que recuerda mucho a la de un "array": $dato["nombre"].
- Finalmente, cerraremos la conexión con la base de datos.

Vamos a ver un ejemplo que recopile todo esto (leer_bd.php):

```
<html>
<body>
<p>Datos extraidos sobre amigos: </p>
<table>
  <tr>
  <td>Nombre</td>
  <td>E-mail</td>
  <td>Tel&eacute;fono</td>
  <td>Fecha Nac.</td>
  </tr>
<?php
if (!($enlace=mysql_connect("localhost","usuario","1234"))) {
    echo "Error conectando a la base de datos.";
    exit();
}

if (!mysql_select_db("prueba",$enlace)) {
    echo "Error seleccionando la base de datos.";
    exit();
}

$resultado=mysql_query("SELECT \* FROM amigos",$enlace);

while($dato = mysql_fetch_array($resultado))   {
    echo "<tr><td>" . $dato["nombre"] . "</td>" .
        "<td>" . $dato["email"] . "</td>" .
        "<td>" . $dato["telefono"] . "</td>" .
        "<td>" . $dato["fechanac"] . "</td>" .
        "</tr>";
}
mysql_free_result($resultado);
mysql_close($enlace);
?>
</table>
</body>
</html>
```

El resultado debería ser el siguiente (los datos que habíamos introducido):

```
Datos extraidos sobre amigos:

Nombre

E-mail

Teléfono

Fecha Nac.

Juan

1111@aprendeaprogramar.com

1111-111-111

2013-11-18

Antonio

2222@aprendeaprogramar.com

2222-222-222

2013-11-19
```

10.3. Inserción de datos en la base de datos

Para introducir datos, normalmente usaremos dos "páginas":

- Una será el formulario con las casillas de introducción, el botón, etc.
- La segunda recibirá los datos del formulario y los insertará en la base de datos (normalmente, antes deberemos asegurarnos de que los datos tienen valores "razonables").

En un proyecto real, en ocasiones será la misma página la que reciba los datos, como ya habíamos comentado, pero de momento vamos a separar la lógica, para simplificar el desarrollo.

La apariencia de la página que pide datos al usuario podría ser:

```
<html>
<body>
<h1>Introducción de datos</h1>
  <form action="almacenar.php" method="post">
    Nombre: <input name="nombre" type="text"> <br> <br>
    Correo electrónico: <input name="email" type="text"> <br> <br>
    Teléfono: <input name="telefono" type="text"> <br> <br>
    Fecha nac.: <input name="fechanac" type="text"> <br> <br>
    <input name="guardar" value="Guardar" type="submit"><br>
</body>
</html>
```

Y la página encargada de guardar estos datos en la base de datos sería muy similar a la que habíamos utilizado para mostrar información extraída de la base de datos, apenas con dos diferencias importantes:

La orden de SQL que habíamos usado para leer era "SELECT FROM"; para introducir datos usaremos "INSERT INTO" (si son datos nuevos) o bien "REPLACE INTO" (si se debe sobrescribir algún dato que pudiera existir previamente y en el que se haya modificado sólo algún campo).

No necesitaremos usar "mysql_fetch_array" para recibir información, ni ningún "while" para que esta recepción de información sea repetitiva.

Podría ser algo así (insisto: en un caso real también se debería comprobar que el usuario realmente ha introducido datos, y que esos datos tienen sentido):

```
<html>
<body>
<?php
$n = $_POST["nombre"];
$e = $_POST["email"];
$t = $_POST["telefono"];
$f = $_POST["fechanac"];

if (!($enlace=mysql_connect("localhost","usuario","1234"))) {
    echo "Error conectando a la base de datos.";
    exit();
}

if (!mysql_select_db("prueba",$enlace)) {
    echo "Error seleccionando la base de datos.";
    exit();
}

mysql_query("REPLACE INTO amigos (nombre, email, telefono,
fechanac) VALUES (".
    "'$n','$e','$t','$f')", $enlace);

mysql_close($enlace);
?>
<p>Datos introducidos: </p>
    Nombre: <?=$n?><br>
    E-mail: <?=$e?><br>
    Tel&eacute;fono: <?=$t?><br>
    Fecha Nac.: <?=$f?><br>
</body>
</html>
```

Una vez introducidos los datos, podríamos volver a mostrarlos de la forma vista en el apartado anterior. La nueva ficha debería aparecer junto a las demás.

Podemos comprobar si los datos se han guardado con "mysql_affected_rows();":

```
$cantidadDeDatos = mysql_affected_rows();
```

Si $cantidadDeDatos vale 0, es que no se ha llegado a guardar ningún dato (típicamente, porque no esté funcionando la conexión a la base de datos o porque ese código ya estuviera utilizado).

11. Fuentes bibliográficas.

Web de Nacho Cabanes: aprende a programar