Aprendiendo PHP

https://fernando-gaitan.com.ar/

Fernando Gonzalo Gaitán

Índice

Al lector

Este documento ha sido creado con fines educativos. Podés copiar y utilizar cualquier parte de su contenido, sin embargo se deberá citar la fuente.

Además el mismo ha sido desarrollado el día 05/01/2014, por este motivo pudiesen existir problemas de incompatibilidad con versiones actuales.

Para seguir en contacto conmigo, podés hacerlo a través de las siguientes redes sociales:

Introducción e instalación

Hasta ahora todas las publicaciones que he subido que tenía entre sus temas PHP lo hice suponiendo que el que lo estaba leyendo ya tenía conocimientos básicos o avanzados de éste. Sin embargo, y aunque hay mucha documentación y tutoriales de PHP, muchas veces se arrancan los temas creyendo de que el que lo está leyendo tiene conocimientos de lo que es programar o de cómo funciona una página web.

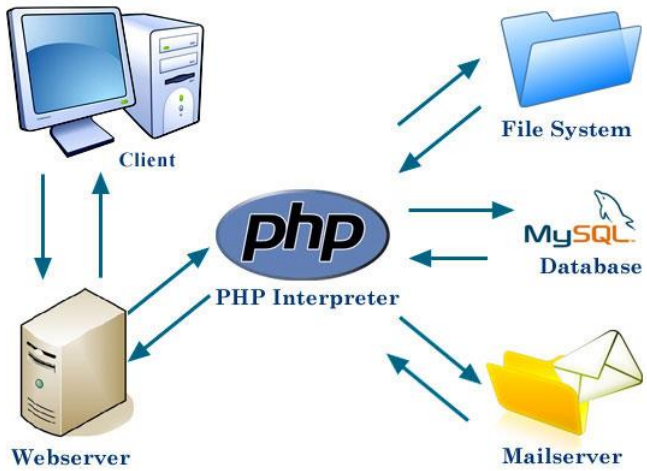
Las siguientes publicaciones que pienso subir las haré suponiendo que el que la está leyendo no sabe absolutamente nada, pero quiere aprender, porque nadie nació sabiendo, pero eso muchas veces la gente que está allá arriba y trata mal a los principiantes que hacen preguntas en foros, no lo tiene en cuenta.

¿Qué es PHP?

La pregunta es muy fácil, aunque puede abrir la puerta a otras preguntas más difíciles que se irán respondiendo con el paso del tiempo. PHP es un lenguaje de programación desde el lado del servidor. Ahora ¿qué es el servidor? Allá vamos…

Seguramente muchos se han preguntado muchas veces cómo funciona una página web, la respuesta es un tanto compleja, pero trataré de responder como decimos en Argentina: ‘En criollo’.

Imaginate que existen dos cosas, algo llamado **cliente** y algo llamado **servidor**. El cliente es un dispositivo, ya sea una PC de escritorio, portatil, un celular, etc; que obviamente se conecta a internet desde un navegador: Chrome, Firefox, Explorer, etc. Por otro lado tenemos el servidor, que es una máquina ubicada físicamente en alguna parte del mundo esperando peticiones para devolver respuestas.



(Fuente de la imagen: https://www.bogotobogo.com/php/images/php1/php_interpreter.jpg)

Osea, me conecto a una página web, yo, el cliente, estoy haciendo una petición al servidor donde se encuentra la página a la que quiero visitar. El servidor procesa mi petición y me devuelve una respuesta. Esa respuesta puede ser por ejemplo el código html que será interpretado por mi navegador, también podría ser un archivo que me quiero descargar. Lleno un formulario y lo envío, estoy enviando también una petición que será procesada también por el servidor, y de nuevo me devolverá algo. Así funciona un ciclo sin fin entre clientes haciendo peticiones y servidores devolviendo respuestas.

Para ser más precisos, el cliente es el usuario que se conecta a una página web, y el servidor es donde se encuentra esa página web.

Instalando PHP en nuestro localhost

Hasta acá todo muy lindo, pero hasta el momento sólo contamos en nuestra máquina con el cliente, tenemos un navegador para hacer peticiones a otras páginas, a otros servidores. Pero no tenemos un servidor para empezar a aprender a programar en PHP. Para contar con un servidor local que nos permita crear nuestros proyectos para luego subirlos a la red vamos a instalar un programa llamado XAMPP que nos instalará también en nuestra máquina Apache, que es el intérprete de PHP, también una base de datos MySQL y otros programas más que vamos a dejar de lado por el momento.

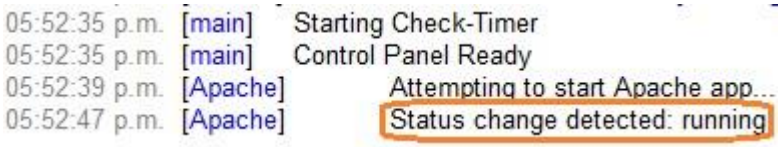
Instalar Xampp

Simplemente selecciona el sistema operativo de tu máquina y descarga el archivo instalable. Una vez descargado, ejecutalo para instalarlo.

Al finalizar la instalación debería aparecer una ventana como está:

Éste es el panel de control de nuestro Xampp. Habrá 5 servicios que podrás iniciar. Vamos a pulsar el botón 'Start' para iniciar nuestro servidor Apache, por el momento ignoremos los otros y dejemos todo quede como está.

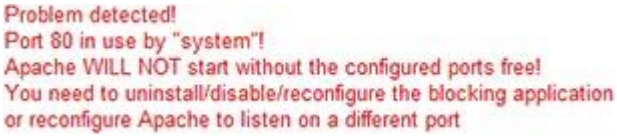
Debería aparecerte un mensaje similar a éste:



Posible error del puerto 80

(Lo que aparece a continuación solamente se debe hacer si tuviste problemas al iniciar el servidor)

El servidor Apache por defecto intenta iniciarse en el puerto 80 de nuestra máquina, pero tal vez tu equipo tenga el puerto 80 ocupado por otra aplicación. Si al intentar iniciar tu apache te aparece por pantalla un mensaje como éste:



Entonces tendrás que modificar el puerto de la siguiente manera:

Primero volvemos al panel de control y pulsamos el botón 'Config' que está a la misma altura de Apache y luego seleccionamos de la lista 'Apache (httpd.conf)'.

Se abrirá un archivo en donde tendrás que buscar la línea:

```
Listen 80
```

Y modificarla por:

```
Listen 81
```

Y además también:

```
ServerName localhost:80
```

Por:

```
ServerName localhost:81
```

Una vez que termines, tendrás que guardar el archivo y volver al panel de control del Xampp. Pulsar en 'Start' nuevamente. Ahora el servidor debería estar listo.

Mi primer PHP

Por empezar para comprobar que todo ha salido bien, vamos a abrir un navegador web cualquier, por ejemplo Google Chrome y escribir lo siguiente como URL.

```
https://localhost
```

Si tuviste que cambiar el puerto 80 por el puerto 81 entonces tendrás que escribir:

```
https://localhost:81
```

Si por pantalla aparece un mensaje similar a éste:

English / Deutsch / Francais / Nederlands / Polski / Italiano / Norwegian / Español / 中文 / Português (Brasil) / 日本語

Entonces todo ha salido bien. Seleccioná tu idioma y listo!

Nada puede malir sal.

"Hola mundo"

Bueno, ahora que ya tenemos instado un servidor Apache, podemos empezar a aprender PHP. En primer lugar, tenemos que tener nuestro servidor iniciado. Si no te acordás cómo era, hay que abrir el panel de control de xampp y pulsar el botón 'Start' correspondiente al servidor Apache.

Bien, a continuación, vamos a descargarnos un programa llamado Notepad, que nos permitirá crear archivos .php. Si bien esto no es necesario, ya que el simple Bloc de notas de Windows alcanza para programar PHP, yo recomiendo para aprender este magnífico programa que es muy liviano y cómodo.

Descargar Notepad.

El Notepad también nos permitirá escribir código en otros lenguajes, como Html, Css, Javascript, Sql, Java, etc, etc, etc.

Una vez descargado vamos a instalarlo. La instalación no debería durar mucho.

Vamos a ir a la carpeta de Xampp que fue creada cuando instalamos dicho programa, en la publicación pasada. Si estás usando Windows 7 como yo lo más probable es que la encuentres en **C:**.

Entramos en **xampp** y vamos adentro de la carpeta **htdocs**. Dentro de la misma crearemos nuestros proyectos. Vamos a empezar con el primero. Creamos una carpeta de nombre **aprendiendo_php**.

Ahora vamos a abrir el Notepad. Por empezar vamos a Lenguaje -> PHP.

Todo lo que se escribe que contenga código PHP debe estar dentro las etiquetas **<?php ?>**:

```
<?php
 //Acá va el código.
?>
```

También podemos escribir comentarios. Los comentarios no se ejecutarán en el script, pero son útiles para que el programador pueda dejar documentado su código.

Los comentarios en una línea se escriben con doble barra:

```
<?php
   //Comentario de una sola línea.
?>
```

O bien, podemos crear comentarios de varias líneas:

```
<?php
   /*
   *Comentario 1
   *Comentario 2
   *Comentario 3
   */
?>
```

Tratar siempre de dejar comentarios, ya que más adelante podemos tener la necesidad nosotros, o algún compañero programador, de editar nuevamente el código y los comentarios pueden ser una buena guía.

Hola mundo

Ok, ahora vamos a pasar a la acción.

Vamos a escribir lo siguiente:

```
<?php
   echo 'Hola mundo';
?>
```

Éste es el típico 'Hola mundo' que suele escribirse al comenzar a programar en un lenguaje de programación, es más una costumbre.

Ahora bien, **echo** es una palabra reservada (ya veremos qué es una palabra reservada) que nos permitirá hacer salidas al navegador. En la publicación pasada dijimos que el servidor se encarga de devolverle respuestas al navegador.

También, al finalizar cada sentencia en PHP, debemos utilizar punto y coma como se ve en el ejemplo.

Además si nos fijamos bien el texto que devolvemos: **hola mundo** se escribe con comillas simples. Esto se debe a que en PHP, como en la mayoría de los lenguajes de programación a la hora de trabajar con cadenas se debe utilizar comillas simples o dobles, para que de esta manera el intérprete comprenda que lo que está dentro de las comillas es un texto.

En PHP tanto las comillas simples como dobles son válidas para las cadenas. Sin embargo, aunque pueden usarse ambas al comenzar con una debe finalizarse con otra. Osea si escribimos un texto que empieza con comillas simple, el mismo debe finalizar también con otra comillas simple, y lo mismo para las comillas dobles.

```
<?php
   echo 'Hola mundo';
?>
```

(FORMA CORRECTA: Empieza con comillas simples, finaliza con comillas simples)

```
<?php
   echo "Hola mundo";
```

```
?>
```

(FORMA CORRECTA: Empieza con comillas dobles, finaliza con comillas dobles)

```
<?php
   echo 'Hola mundo";
?>
```

(FORMA INCORRECTA: Empieza con comillas simples, finaliza con comillas dobles)

```
<?php
   echo "Hola mundo';
?>
```

(FORMA INCORRECTA: Empieza con comillas dobles, finaliza con comillas simples)

```
<?php
   echo Hola mundo;
?>
```

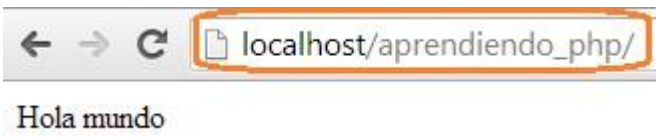
(FORMA INCORRECTA: El texto no está entre comillas)

Bueno, ahora vamos a probar que lo que acabamos de escribir funciona correctamente.

En primer lugar vamos a guardar el archivo con el código que acabamos de escribir, (el 'hola mundo' con comillas simples) Vamos a Archivo -> Guardar y lo guardamos con el nombre **index.php**. Recordar no olvidarse el .php ya que es la extensión de este tipo de archivos. Lo guardamos dentro de la carpeta que creamos anteriormente: **aprendiendo_php**. Si no te acordás estaba dentro de xampp -> htdocs -> aprendiendo_php.

Ahora vamos a abrir nuestro navegador y a escribir la siguiente dirección:

```
localhost/aprendiendo_php/
```



Recordar que si tenés configurado el servidor apache en otro puerto, por ejemplo el 81, hay que ingresar después de localhost dos puntos y el puerto:

```
localhost:81/aprendiendo_php/
```

Si apareció por pantalla el mensaje:

```
Hola mundo
```

Todo ha salido bien.

Ahora, por qué decidimos ponerle a nuestro archivo de nombre index.php. En la configuración por defecto de nuestro apache, al ingresar en un directorio sin aclararle el archivo .php buscará index.php. También nosotros podríamos ingresar con la direccción:

```
localhost/aprendiendo_php/index.php
```

Y nos daría el mismo resultado.

Ahora vamos a hacer otra prueba. Vamos a cerrar el archivo que acabamos de crear y vamos a crear un nuevo archivo al que le vamos a escribir lo siguiente:

```
<?php
   echo 'Hola. Soy Fernando.';
   echo '<br />';
   echo 'Estoy aprendiendo PHP.'
?>
```

Y lo vamos a guardar como **saludar.php**. Siempre guardándolo en xampp -> htdocs -> aprendiendo_php. Abrimos el navegador e ingresamos lo siguiente:

```
localhost/aprendiendo_php/saludar.php
```

Si te fijas bien, en cada instrucción debemos finalizar con punto y coma como lo dijimos antes, de lo contrario nos devolverá error. A su vez podemos utilizar código html dentro de cada cadena, en este caso usamos la etiqueta **br** para hacer un salto de línea, aunque esto último no se recomienda,.

Bien con esto terminamos por hoy. Ya sabemos la forma en la que trabaja las rutas con Apache y escribimos nuestra primer práctica en PHP.

Saludos!!!

Variables

En todos los lenguajes de programación siempre se requiere la posibilidad de guardar datos temporales para realizar distintas operaciones. En PHP esto se conoce como variables. Las variables nos permiten guardar datos dentro de cada script.

Formato de una variable

Para crear una variable en PHP debemos usar el signo **$** como primer caracter seguido del signo nombre de la variable, el caracter = (igual) y el valor de la variable:

```
<?php
   $nombre = 'Fernando';
?>
```

Además las variables deben estar formadas por los siguientes caracteres:

- Letras de la A a la Z. Pueden ser minúsculas o mayúsculas, pero no se pueden utilizar caracteres especiales como tildes, ñ, etc.
- Números. Estos también están permitidos, sin embargo no pueden utilizarse como primer caracter luego del signo pesos. Después del signo pesos solamente pueden ir caracteres alfabéticos o guiones bajo.
- Guiones bajo. Estos caracteres también están permitidos.

Ejemplos de formas correctas e incorrectas de escribir una variable:

```
$miVariable_1 = 'Valor';
```

(FORMA CORRECTA: Después del signo $ hay un caracter alfabético)

```
$2miVariable_1 = 'Valor';
```

(FORMA INCORRECTA: Después del signo $ hay un caracter numérico)

```
$_miVariable_1 = 'Valor';
```

(FORMA CORRECTA: Después del signo $ hay un guión bajo)

```
$MiVariable_1 = 'Valor';
```

(FORMA CORRECTA: Después del signo $ hay un caracter alfábetico, aunque es en mayúscula, esto también está permitido)

```
$MiVariable_1.ñ = 'Valor';
```

(FORMA INCORRECTA: Hay caracteres no permitidos, la ñ y el punto)

```
miVariable_1 = 'Valor';
```

(FORMA INCORRECTA: Las variables siempre deben comenzar con un signo pesos, que acá brilla por su ausencia)

Case sensitive

Dijimos que las variables pueden contener caracteres alfabéticos tanto en minúscula como en mayúscula, pero OJO. Hay un error típico que hasta programadores experimentados suelen errar. Para PHP una **a** y **A** no es lo mismo. Osea, podemos hacer esto:

```
<?php
   $nombre = 'Fernando';
   echo $nombre;
```

```
?>
```

Creamos una variable llamada **$nombre** de valor 'Fernando'. Luego la imprimimos por pantalla. Hasta acá todo bien. Sin embargo nosotros no podemos hacer esto por ejemplo:

```
<?php
   $nombre = 'Fernando';
   echo $NOMBRE;
?>
```

Para PHP **$nombre** y **$NOMBRE** son variables diferentes, por tanto al imprimirlo por pantalla me lanzará un error ya que estamos imprimiendo una variable que no tiene valor.

Sobreescribir variables

Las variables, como lo indica su nombre, pueden variar o modificar su valor. Por tanto, si nosotros hacemos por ejemplo esto:

```
<?php
   $nombre = 'Fernando';
   $nombre = 'Gonzalo';
   echo $nombre;
?>
```

El resultado que nos devolverá por pantalla es:

```
Gonzalo
```

Debido a que, si bien la variable inicia con un valor, 'Fernando', el último que guardamos es otro, 'Gonzalo'.

Otro ejemplo práctico puede ser:

```
<?php
   $nombre = 'Fernando';
   echo $nombre;
   echo '<br />';
   $nombre = 'Gonzalo';
   echo $nombre;
   echo '<br />';
?>
```

En este último caso, el resultado será:

```
Fernando
Gonzalo
```

Osea, primero guardamos la variable **$nombre** con el valor 'Fernando', imprimimos por pantalla el valor de la misma y un salto de línea, imprime ese valor, porque es el valor actual. Pero en la línea siguiente la variable cambia su valor por 'Gonzalo' y al imprimir nuevamente la misma por pantalla el valor es el nuevo, el valor actual, 'Gonzalo'.

Tipos de datos

Ahora bien, en PHP a diferencia de otros lenguajes de programación, no debemos especificar qué tipo de datos debemos guardar, sin embargo es importante conocer cuáles son estos.

Hasta ahora sólo hemos visto guardar cadenas en una variable, sin embargo no es el único tipo de dato.

```
<?php
   $nombre = 'Fernando';
   $edad = 27;
   $programador = true;
?>
```

Los tipos de datos pueden ser:

- **String**: Son cadenas; una letra, una palabra, una frase, la letra de una canción. Siempre entre comillas dobles o simples.
- **Integer**: Son números enteros, osea sin coma. Por ejemplo para guardar un año, una edad. También pueden ser positivos, negativos y cero.
- **Float**: Son números decimales, números con coma. Por ejemplo para guardar el precio de un producto. También pueden ser positivos, negativos y cero.
- **Boolean**: Son valores lógicos, osea **true** (Verdadero) o **false** (Falso)
- **Array**: Se usan para guardar una colección de datos.
- **Object / Class**: Son tipos de datos más complejos. Los veremos mucho más adelante.
- **Null**: Es un dato que aun no tiene valor.
- **Unknow type**: Tipo de dato desconocido.

Números

En la publicación anterior vimos para qué sirven las variables y además aprendimos que cada una de ellas posee un tipo de dato, dependiendo del valor que se guarda. En esta ocasión veremos las variables numéricas, osea **integer** (enteros) y **float** (decimales), además éstas pueden ser positivos, negativos y cero.

Operaciones aritméticas

En PHP realizar sumas, restas, multiplicaciones y divisiones es muy fácil. Por ejemplo:

```
<?php
   $resultado = 10 + 5 * 2;
   echo $resultado;
```

```
?>
```

Esto nos devolverá por pantalla:

```
20
```

En primer lugar se calcula el producto, 5 * 2, osea 10. Luego este resultado se le sumará 10, entonces será 10 + 10, osea 20.

Pero si en cambio lo que nosotros queremos es que primero se realice la suma, 10 + 5 y el resultado de la mismo se multiplique por 2, entonces deberíamos separar por paréntesis:

```
<?php
   $resultado = (10 + 5) * 2;
   echo $resultado;
?>
```

En este caso el resultado que devolverá por pantalla será:

```
30
```

Los paréntesis nos permitirán separar las distintas partes de nuestra operación matemática. Otro ejemplo:

```
<?php
   $resultado = ((10 + 5) * 2) - (50 / 5);
   echo $resultado;
?>
```

Primero se calcula la sumatoria de 10 + 5, que da como resultado 15 y al multiplicarla por 2, **30**. Desde el otro lado dividimos 50 por 5, osea **10.** Y al restar 30 menos 10, el resultado es el que nos arroja por pantalla:

```
20
```

Números decimales

En PHP podemos trabajar con decimales. Algo importante es que al escribir lo que nosotros llamamos coma, aquí se escribe con un punto, ya que PHP usan la forma de escribir decimales al igual que lo hacen en los Estados Unidos.

```
<?php
   $resultado = 40.25 + 20.70;
   echo $resultado;
?>
```

El resultado naturalmente sería:

```
60.95
```

Números negativos

En PHP también existen los números negativos, que como en la vida real deben escribirse con el signo menos (-) adelante para distinguirlos:

```
<?php
   $numero1 = -10;
   $numero2 = -5;
   $resultado = $numero1 + $numero2;
   echo $resultado;
?>
```

Aquí creamos dos variables con números negativos, -10 y -5, así que al sumarlas nos da -15.

Incremento y decremento

Muchas veces podemos tener la necesidad de sumar o restar el valor de una variable para llevar un control de las modificaciones que va sufriendo un número.

Por ejemplo, si tuviéramos una variable que guardara un número, un 1 (uno) y queremos sumarle también un 1 podríamos hacerlo mediante a la sobrescritura de la variable:

```
<?php
   $numero = 1;
   $numero = $numero + 1;
   echo $numero;
?>
```

En este caso, la variable **$numero** comienza con el valor 1, luego la sobrescribimos igualando el valor actual + 1 y al imprimirla por pantalla el resultado es obviamente:

```
2
```

Aunque esto puede solucionarse de forma más amigable gracias al operado de incremento:

```
<?php
   $numero = 1;
   $numero++;
   echo $numero;
?>
```

Con tan sólo agregar el **++** al final de la variable, la misma sumará un número más. También podríamos hacerlo con el signo ++ pero al principio.

```
<?php
```

```
   $numero = 1;
   ++$numero;
   echo $numero;
?>
```

Esto lograría el mismo resultado.

La primer forma, la de utilizar el ++ al final se llama Post-Incremento, mientras que ésta última se conoce como Pre-Incremento. La diferencia entre ambas es que la primera devuelve por pantalla el resultado actual y luego la suma, mientras que la segunda devuelve el valor con la suma incluida. Por ejemplo:

Esto:

```
<?php
   $numero = 1;
   echo ++$numero;
?>
```

Devuelve:

```
2
```

La razón es que PHP lee de izquierda a derecha, por tanto, al leer los signos ++ antes de la variable va a imprimir la sumatoria final. Mientras que si hacemos esto:

```
<?php
   $numero = 1;
   echo $numero++;
?>
```

El resullado sería:

```
1
```

Ya que el ++ está al final, el valor que devolverá por pantalla es el actual, antes de sumar, osea el 1.

A través del ++ la sumatoria se realizará de a 1. Por ejemplo:

```
<?php
   $numero = 1; //La variable comienza con 1.
   $numero++; //Se suma 1, la variable ahora es 2.
   $numero++; //Se suma 1, la variable ahora es 3.
   $numero++; //Se suma 1, la variable ahora es 4.
   echo $numero; //Se imprime por pantalla el valor actual, osea 4.
?>
```

En este último caso la variable comienza con 1, pero se suma 3 veces, así que el resultado finalmente es 4.

Pero la única forma de incrementar su valor no es sólo de a 1, también podemos hacerlo de a 2, 5, 10, 1000 o lo que queramos con **+=**. Por ejemplo si quisiéramos ir desde el 10 al 25:

```
<?php
   $numero = 10; //La variable comienza con 10.
   $numero+=5; //Se suma 5, la variable ahora es 15.
   $numero+=5; //Se suma 5, la variable ahora es 20.
   $numero+=5; //Se suma 5, la variable ahora es 25.
   echo $numero; //Se imprime por pantalla el valor actual, osea 25.
?>
```

En este caso empezamos con el valor 10, pero el mismo se va sumando de a 5.

Todo esto que vimos hasta ahora también se aplica para restar, a través del operador de decremento. Por ejemplo:

```
<?php
   $numero = 5;
   $numero--;
   $numero--;
   $numero--;
   $numero--;
   $numero--;
   echo $numero;
?>
```

En este último caso usamos el operador — restando de a 1. El número comienza en 5, pero finalmente termina en 0.

Y lo mismo para hacerlo de a muchos:

```
<?php
   $numero = 1000;
   $numero -= 500;
   echo $numero;
?>
```

Aquí comenzamos con el número 1000, pero le restamos de a 500 y el resultado finalmente es 500.

Cadenas de texto

En la publicación pasada aprendimos a trabajar con las variables numéricas, en esta ocasión aprenderemos lo que es probablemente el tipo de dato más utilizado en PHP, los **string** o cadenas de texto.

En primer lugar para definir el valor de un string, debemos utilizar comillas simples o dobles.

Comillas simples:

```
<?php
   $saludo = 'Hola mundo';
?>
```

Comillas dobles:

```
<?php
   $saludo = "Hola mundo";
?>
```

Concatenación

El concatenado nos permite unir cadenas a través del operador punto (.) Por ejemplo:

```
<?php
   $nombre = 'Fernando';
   $apellido = 'Gaitan';
   $nombre_completo = $nombre . ' ' . $apellido;
   echo $nombre_completo;
?>
```

Esto finalmente nos devolverá por pantalla:

```
Fernando Gaitan
```

También podríamos concatenar una cadena dentro de la misma variable:

```
<?php
   $nombre_completo = 'Fernando';
   $nombre_completo = $nombre_completo . ' Gaitan';
   echo $nombre_completo;
?>
```

En este caso creamos una variable con un valor, y luego la sobreescribimos concatenando el valor actual de la variable con el texto adicionado y devolvería lo mismo.

Asignación

Otra forma de unir cadenas es mediante el operador de asignación punto igual (.=)

```
<?php
 $cancion = 'Cuenta la historia de un mago';
 $cancion .= ' / ';
 $cancion .= 'Que un día en su bosque encantado lloró';
 $cancion .= ' / ';
```

```
 $cancion .= 'Porque a pesar de su magia no había podido encontrar el
amor.';
 echo $cancion;
?>
```

Esto imprimirá por pantalla:

```
Cuenta la historia de un mago / Que un día en su bosque encantado lloró /
Porque a pesar de su magia no había podido encontrar el amor.
```

Mediante el operador de asignación iremos, como su palabra lo indica, asignando o agregando texto a una variable de tipo string.

Comillas simples y dobles

A pesar de que podemos usar comillas simples y dobles para definir cadenas de texto, ambas formas presentan una diferencia. Si nosotros tuviésemos por ejemplo que concatenar un string con otras variables tendríamos que hacer algo cómo esto:

```
<?php
   $nombre = 'Charly';
   $color = 'Blanco';
   $raza = 'Dogo';
   echo $nombre . ' es un perro de color ' . $color . ' y de raza ' .
$raza;
?>
```

Esto imprime por pantalla:

```
Charly es un perro de color Blanco y de raza Dogo
```

Al tener una cadena que mezcla textos con variables debemos concatenar todo separando con puntos el texto y las variables.

Sin embargo, si hiciéramos lo mismo con comillas dobles no necesitaríamos utilizar el operador punto para concatenar, ya que las comillas dobles pueden distinguir dentro de sí cuando llamamos a una variable. Entonces otra forma más amigable de realizar lo que hicimos es:

```
<?php
   $nombre = 'Charly';
   $color = 'Blanco';
   $raza = 'Dogo';
   echo "$nombre es un perro de color $color y de raza $raza";
?>
```

Por esto siempre se aconseja que las comillas dobles se utilicen sólo cuando la cadena está mezclada con texto, en el resto de los casos, se debería usar comillas simples.

Otra razón para saber si usamos una u otra es dependiendo de si el texto muestra, literalmente comillas simples o dobles. Por ejemplo para mostrar por pantalla las comillas dobles:

```
<?php
   $frase = 'Arquimedes gritaba: "Eureka, Eureka" mientras celebraba su
descubrimiento.';
   echo $frase;
?>
```

Esto nos devolverá por pantalla:

```
Arquimedes gritaba: "Eureka, Eureka" mientras celebraba su
descubrimiento.
```

Si en cambio necesitamos usar comillas simples:

```
<?php
   $frase = "Arquimedes gritaba: 'Eureka, Eureka' mientras celebraba su
descubrimiento.";
   echo $frase;
?>
```

Esto en cambio devolverá:

```
Arquimedes gritaba: 'Eureka, Eureka' mientras celebraba su
descubrimiento.
```

Arreglos

Hasta ahora hemos aprendido que las variables en PHP nos permiten guardar valores, como cadenas de texto, números o valores lógicos (true o false), pero también existe un tipo de dato que nos permite guardar una colección de estos.

Por ejemplo, nosotros podríamos tener un valor de tipo **integer** para guardar un número:

```
<?php
   $numero = 5;
?>
```

Pero si por ejemplo tuviésemos que guardar los 6 números que salieron en la lotería, ¿entonces necesitaríamos 6 variables?, ¿y si tuviéramos 100 números, 100 variables? Bueno, no, para guardar varios datos debemos usar el tipo de dato **array**:

```
<?php
   $numeros = array(38 , 36 , 6 , 2 , 16 , 19);
?>
```

Como se ve en el ejemplo, para guardar un array o arreglo en una variable debemos utilizar la palabra reservada array y los valores separados por coma.

Guardar y recuperar datos de un arreglo

Para guardar un valor nuevo en un array debemos llamar al nombre de la variable con corchetes (**[]**) y asignarle el nuevo valor. Por ejemplo podríamos crear un array vacío:

```
<?php
   $articulos = array();
?>
```

Aquí tenemos nuestro array vacío, y ahora podemos agregarle nuevas posiciones:

```
<?php
   $articulos = array();
   $articulos[] = 'Birome';
   $articulos[] = 'Cuaderno';
   $articulos[] = 'Resaltador';
   $articulos[] = 'Crayones';
   $articulos[] = 'Corrector';
?>
```

Y luego recuperar el valor de cualquiera de estos llamándolo por su posición, empezando por cero. Por ejemplo si quisiéramos recuperar el primero valor (birome):

```
<?php
   $articulos = array();
   $articulos[] = 'Birome';
   $articulos[] = 'Cuaderno';
   $articulos[] = 'Resaltador';
   $articulos[] = 'Crayones';
   $articulos[] = 'Corrector';
   echo $articulos[0];//Devuelve el valor Birome.
?>
```

A través de los corchetes vacíos nosotros agregaremos una última posición al array. Pero también nosotros podemos setear una posición X. Por ejemplo:

```
<?php
   $frutas = array();
   $frutas[0] = 'Banana';
   $frutas[1] = 'Manzana';
   $frutas[2] = 'Naranja';
   echo $frutas[1];//Devuelve el valor Manzana.
?>
```

Arreglos asociativos

Otra forma de crear arreglos que puede servirnos cuando contamos con datos que se distinguen entre sí, son los arrays asociativos. Por ejemplo si quisiéramos guardar los datos de alguien:

```
<?php
   $persona = array(
      'nombre' => 'Fernando',
      'apellido' => 'Gaitan',
      'edad' => 27,
      'programador' => true
   );
?>
```

Y luego para acceder a una de sus posiciones deberíamos usar corchetes con lo que se conoce como índice:

```
<?php
   $persona = array(
      'nombre' => 'Fernando',
      'apellido' => 'Gaitan',
      'edad' => 27,
      'programador' => true
   );
   echo $persona['nombre'];//Devuelve Fernando.
?>
```

Arreglos multidimensionales

Bien, sabemos que un array puede guardar cadenas, números y valores de tipo boolean. Pero un array en realidad puede guardar una colección de cualquier tipo de dato como otro array o un objeto (mucho más adelante veremos que es un objeto) Si quisiéramos hacer lo mismo que hicimos en el anterior ejemplo, pero guardando un lista con varias personas entonces deberíamos hacer algo como esto:

```
<?php
   $personas = array(
   array(
      'nombre' => 'Fernando',
      'apellido' => 'Gaitan',
      'edad' => 27,
      'programador' => true
   ),
   array(
      'nombre' => 'Tomás',
      'apellido' => 'Benvenuto',
      'edad' => 20,
      'programador' => true
   ),
```

```
   array(
      'nombre' => 'Andrés',
      'apellido' => 'Martínez',
      'edad' => 32,
      'programador' => false
   )
 );
?>
```

Y para recuperar el valor deberíamos llamar a la posición y el índice. Por ejemplo, si quisiéramos devolver la edad de la segunda persona:

```
echo $personas[1]['edad'];//Esto devolverá 20
```

Constantes

En PHP una variable no sólo permite guardar datos, a su vez esos datos pueden ser modificados:

```
<?php
   $color = 'Amarillo';
   $color = 'Rojo';
   $color = 'Azul';
?>
```

Aquí creamos una variable que inicialmente tiene el valor ‘Amarillo’, luego la modificamos por ‘Rojo’ y finalmente por ‘Azul’. Pero ahora el asunto es que muchas veces necesitamos guardar datos que no deberían ser modificados nunca.

Las contantes también permiten guardar datos, pero a diferencia de las variables su valor no puede ser modificado a lo largo de nuestra aplicación. Por ejemplo la url de nuestro sitio:

```
<?php
 define('URL_SITIO', 'https://fernando-gaitan.com.ar/');
 echo URL_SITIO;
?>
```

Para crear una constante debemos utilizar la palabra define e ingresar en primer lugar entre comillas el nombre de la constante y en segundo lugar el valor. Los nombres de las constantes deben escribirse con letras mayúsculas y guiones bajo. Luego como se muestra en el ejemplo, para recuperar el valor de una constante debe llamarse a la misma, pero sin las comillas.

Otro ejemplo puede ser también PI, que siempre es 3.14:

```
<?php
   define('PI', 3.14);
   echo PI;
?>
```

Una cosa que hay que tener en cuenta es que las constantes pueden guardar datos de tipo string, number y boolean, sin embargo no se pueden guardar datos más complejos como arreglos u objetos.

Condicionales

En todos los lenguajes de programación existe algo llamado condicionales, estos permite controlar el flujo de nuestra aplicación, lo que significa que el comportamiento del mismo dependerá de determinadas condiciones.

Vamos a transportar esto a un ejemplo de la vida real. Supongamos que hay una película que es apta mayores de 16 años, por tanto las personas podrán ingresar a la sala a ver la película si cumplen con una condición: ser mayores de 16 años. Si la persona cumple con esa condición podrá ingresar a ver la película, sino tendrá que irse.

if else

El condicional más utilizado es el if (si) else (sino). La sintaxis es muy sencilla:

```
<?php
   if($condicion){
      //Ejecutará esto cuando la condición se cumpla.
   }else{
      //Ejecutará esto cuando la condición NO se cumpla.
   }
?>
```

Esto está formado por tres partes. La primera es la condición, si la misma se cumple entrará por el lado verdadero, sino por el lado falso, osea no se cumple la condición. Tanto si se cumple o no, el código que se ejecutará debe estar dentro de llaves.

Por ejemplo lo que acabamos de decir:

```
<?php
   $mayor_de_edad = true;
   if($mayor_de_edad){
      echo 'Usted puede ver la película';
   }else{
      echo 'Usted NO puede ver la película';
   }
?>
```

En este caso, se ejecutará el lado verdadero, donde imprime por pantalla: ‘Usted puede ver la película’, ya que el valor de **$mayor_de_edad** es **true**. Podés probar cambiando esta variable a **false**, para comprobar que sale por el otro lado.

También podemos omitir la parte del else:

```
<?php
   $mayor_de_edad = true;
   if($mayor_de_edad){
      echo 'Usted puede ver la película';
   }
?>
```

Operador de Negación

También podemos preguntar al revés, osea en lugar de si se cumple la condición, si la misma no se cumple, esto mediante el operador de negación (!):

```
<?php
   $mayor_de_edad = true;
   if(!$mayor_de_edad){
      echo 'Usted NO puede ver la película';
   }else{
      echo 'Usted puede ver la película';
   }
?>
```

Operadores And y Or

El if tiene la capacidad de analizar si se cumple una condición, o bien, una condición formada por más de una, por tanto, podemos preguntar por ejemplo que se cumplan dos condiciones mediante el operador **and**:

```
<?php
   $bateria_cargada1 = true;
   $bateria_cargada2 = true;
   if($bateria_cargada1 and $bateria_cargada2){
      echo 'El control funciona correctamente.';
   }else{
      echo 'El control no funciona debido a que una o ambas baterías no
están cargadas';
   }
?>
```

En este caso el condicional preguntará si tanto **$bateria_cargada1** y **$bateria_cargada2** son **true**, en ese caso se cumplirá la condición, pero en caso contrario, osea que una o ambas variables sean **false**, la condición no se cumplirá.

El operador **or**, en cambio, evaluará que al menos una de las condiciones se cumpla

```
<?php
```

```
   $conexion1 = false;
   $conexion2 = true;
   if($conexion1 or $conexion2){
      echo 'Sistema conectado correctamente';
   }else{
      echo 'El sistema no puede conectarse debido a que ninguna de las
conexiones están funcionando';
   }
?>
```

En este último caso la condición se cumplirá si una de las dos variables, **$conexion1** o **$conexion2**, son **true**, sólo si ambas son **false** no se cumplirá.

Operadores Igual o Diferente

También podemos evaluar que dos valores sean iguales o no lo sean, por ejemplo:

```
<?php
   $nombre = 'Fernando';
   if($nombre == 'Fernando'){
      echo 'Bienvenido Fernando.';
   }else{
      echo 'Bienvenido anónimo.';
   }
?>
```

En esta caso, la condición es que la variable **$nombre** tenga el valor de 'Fernando', si el valor es ése entonces la condición se cumplirá, pero si es otro nombre, por ejemplo 'Juan', entonces se ejecutará la otra parte, la del else.

Ojo, no hay que confundir el =(igual) con ==(igual igual), = se utiliza para asignar un valor a una variable, mientras que == es un operador que evaluará que un valor sea igual a otro.

Pero como tenemos un operador de igual, también tenemos uno de diferente:

```
<?php
   $nombre = 'Fernando';
   if($nombre != 'Fernando'){
      echo 'Bienvenido anónimo.';
   }else{
      echo 'Bienvenido Fernando.';
   }
?>
```

En este último caso preguntamos por diferente, en lugar de igual. Si el nombre no es 'Fernando' la condición se cumplirá, de lo contrario saldrá por el otro lado.

Operadores Idéntico y No idéntico

Ambos ejemplos **igual** o **diferente** también se aplican para los números, sin embargo, tener en cuenta que para PHP:

```
<?php
   $numero = '7';
?>
```

Es igual a:

```
<?php
   $numero = 7;
?>
```

Por tanto si preguntamos:

```
<?php
   $numero1 = '7';
   $numero2 = 7;
   if($numero1 == $numero2){
      echo 'Los numeros son iguales';
   }else{
      echo 'Los numeros NO son iguales';
   }
?>
```

En este caso, la condición se cumple ya que ambos son 7, a pesar de que las variables son de tipos de datos diferentes.

Pero si en cambio queremos evaluar que los valores sean igual y además también lo sean los tipos de datos, debemos usar el operador idéntico, mediante el triple igual (===):

```
<?php
   $numero1 = '7';
   $numero2 = 7;
   if($numero1 === $numero2){
      echo 'Los números y tipos son iguales';
   }else{
      echo 'Los números NO son iguales';
   }
?>
```

O bien, usar el operador de no idéntico:

```
<?php
   $numero1 = '7';
   $numero2 = 7;
   if($numero1 !== $numero2){
      echo 'Los números y tipos NO son iguales';
   }else{
      echo 'Los números y tipos NO son iguales';
   }
?>
```

Operadores mayor o menor

Así como podemos comparar que un número sea igual a otro, también podemos avaluar que un valor sea mayor, menor, mayor igual o menor igual:

- **($valor1 < $valor2)** Menor que
- **($valor1 > $valor2)** Mayor que
- **($valor1 <= $valor2)** Menor igual que
- **($valor1 >= $valor2)** Mayor igual que

Por ejemplo podríamos verificar si la persona tiene edad para ver la película, como en el ejemplo del principio:

```
<?php
   $edad = 17;
   if($edad >= 16){
      echo 'Usted puede ver la película';
   }else{
      echo 'Usted NO puede ver la película';
   }
?>
```

En el ejemplo preguntamos si la edad es mayor o igual a 16, entonces la persona puede ver la película, como es 17 el valor se cumple la condición.

else if

Los if no sólo pueden evaluar si se cumple una condición, pueden hacerse con dos, tres, cuatro o las que fueran necesarias. Por ejemplo, si tuviésemos un número y tuviésemos que evaluar si el mismo es positivo, negativo o cero, tendríamos que preguntar primero si el número es mayor a 0, osea positivo; luego si es menor a 0, negativo; y si ninguna de las condiciones se cumple es porque el número, claro está, es 0. Para eso deberíamos hacer lo siguiente:

```
<?php
   $numero = -10;
   if($numero > 0){
      echo 'El número es positivo.';
   }else if($numero < 0){
      echo 'El número es negativo.';
   }else{
      echo 'El número es cero.';
   }
?>
```

En este caso no se cumple la primer condición, el número no es positivo, pero sí se cumple la segunda, éste es negativo.

Swtich

Muchas veces debemos evaluar el valor de algo ante varias posibilidades, por ejemplo el típico disquito molesto que muchos odiamos cuando llamamos a un lugar y nos da varias opciones para marcar con nuestro teléfono. Si tuviésemos que usar un if, deberíamos hacer algo como esto:

```
<?php
   $numero = 3;
   if($numero == 1){
       echo 'Transferido a servicio técnico';
   }else if($numero == 2){
      echo 'Transferido a ventas';
   }else if($numero == 3){
      echo 'Transferido a consultas';
   }else if($numero == 4){
      echo 'Transferido a reclamos';
   }else{
      echo 'El código ingresado es incorrecto';
   }
?>
```

En este caso en las distintas condiciones siempre preguntamos por el valor de la variable **$numero**, si es igual a 1, 2, 3, 4 o si no es ninguna de estas.

Una forma más amigable de evaluar varias posibilidades con el mismo dato es a través del **switch**. Podríamos reemplazar esto último por esto:

```
<?php
   $numero = 3;
   switch ($numero) {
      case 1:
         echo 'Transferido a servicio técnico';
         break;
      case 2:
         echo 'Transferido a ventas';
         break;
      case 3:
         echo 'Transferido a consultas';
         break;
      case 4:
         echo 'Transferido a reclamos';
         break;
      default:
         echo 'El código ingresado es incorrecto';
   }
?>
```

Switch evaluará el valor de algo específico, en este caso la variable **$numero**. Luego debemos ingresar a través de la palabra reservada **case** qué debe hacerse dependiendo del valor, en cuanto a la palabra **break** permite finalizar el condicional para no seguir preguntando por los case que se encuentren debajo.

Bucles

Los bucles, también conocidos como loops o ciclos de repetición, se utilizan en programación para repetir una cierta cantidad de veces un bloque de código para así lograr resultados como por ejemplo recorrer una lista.

While

El bucle **while** evaluará una condición y de cumplirse realizará una operación que se repetirá tantas veces mientras que se siga cumpliendo la misma.

Por ejemplo vamos a sumar una variable numérica tantas veces mientras que ésta sea menor a 10, empezando por 0:

```
<?php
   $contador = 1;
   $limite = 10;
   while($contador < $limite){
      echo $contador . '<br />';
      $contador++;
   }
?>
```

Esto imprimirá por pantalla:

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

Ahora vamos a analizar el código, lo primero que evalúa el **while** es que se cumpla una condición que **$contador** (que vale por empezar 1) sea menor a **$limite** (que vale 10) De cumplirse esta condición se imprime por pantalla el valor de **$contador**, y luego se la incrementa un número más. Pensemos cómo actúa el intérprete, **$contador** vale 1 ¿Es menor que **$limite** que vale 10? Sí, así que entra, imprime **$contador** con un salto de línea, que es 1 y la incrementa uno más, entonces ahora **$contador** vale 2. Vuelve a preguntar si se cumple la condición **$contador**, que ahora vale 2, ¿es menor a $limite que vale 10? Sí, vuelve a entrar. Y así lo hará hasta que **$contador** deje de ser menor a **$limite**, por ende la condición no se cumplirá y el intérprete seguirá de largo.

Do while

El bucle **do while** funciona casi igual que el **while**, con la diferencia de que **while** primero pregunta si se cumple la condición y de ser así ejecuta el bloque de código, mientras que **do while** ejecuta el bloque y luego pregunta.

Por ejemplo supongamos que tenemos un código como el anterior, pero en donde la variable **$contador** sea 10, igual que **$limite**:

```
<?php
   $contador = 10;
   $limite = 10;
   while($contador < $limite){
      echo $contador . '<br />';
      $contador++;
   }
?>
```

Esto no imprime nada por pantalla ¿Por qué? Muy sencillo. **$contador** tiene el valor 10 y **$limite** también tiene el valor 10, primero se verifica que se cumpla la condición. ¿**$contador** es menor a **$limite**? No, entonces nunca entra el bloque de código a repetir.

Pero si en cambio hacemos esto:

```
<?php
   $contador = 10;
   $limite = 10;
   do {
      echo $contador . '<br />';
      $contador++;
   }while($contador < $limite);
?>
```

Esto imprime por pantalla:

```
10
```

¿Por qué? Porque el **do while** primero ejecuta el bloque de código, osea que imprime por pantalla el valor de **$contador**, que es 10 y lo incrementa uno más, osea 11. Luego sí, pregunta si **$contador** es menor a **$limite**, en nuestro ejemplo si 11 es menor a 10. ¿Lo es? No. Entonces aquí finaliza el ciclo.

For

Este bucle permite repetir un bloque de código desde un punto hasta otro, por ejemplo nosotros podemos imprimir por pantalla números desde el 1 al 10 cómo hicimos con el **while**, pero sin una condición:

```
<?php
   for($i = 1; $i < 10; $i++){
      echo $i . '<br />';
   }
?>
```

Esto imprimirá por pantalla:

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

El for está compuesto por tres partes. En primer lugar creamos una variable llamada **$i**, con un valor inicial, en nuestro caso 1. Luego le indicamos hasta qué punto el **for** se repetirá, como muestra el ejemplo hasta el 10, y por último de a cuanto incrementaremos el valor de la variable **$i**.

También podríamos imprimir los números hasta el 10 incluido cambiando:

```
$i < 10
```

Por:

```
$i <= 10
```

Otra cosa que puede hacerse es que el bucle se repita de mayor a menor:

```
<?php
   for($i = 9; $i > 0; $i--){
      echo $i . '<br />';
   }
?>
```

Esto nos devolverá:

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

Y para incrementar no sólo podemos hacerlo de a 1, con ++, también podemos hacerlo, por ejemplo de a 10:

```
<?php
   for($i = 0; $i < 100; $i+=10){
      echo $i . '<br />';
   }
?>
```

Esto va a imprimir por pantalla:

```
10
20
30
40
50
60
70
80
90
```

Una utilidad que también puede tener un **for** es para recorrer un array:

```
$nombres = array('Juan', 'Pedro', 'Maria');
```

Deberíamos empezar por la primer posición, que es siempre 0 y luego deberíamos contar 3, que es la cantidad que tiene el array. Para saber la cantidad de posiciones que tiene un array debemos usar la función **count()** (ya veremos pronto que son las funciones):

```
$cantidad_de_nombres = count($nombres); //Esto devolverá 3.
```

Así que con esto nos alcanzaría para recorrer este array con un **for**:

```
<?php
   $nombres = array('Juan', 'Pedro', 'Maria');
   $cantidad_de_nombres = count($nombres); //Esto devolverá 3.
   for($i = 0; $i < $cantidad_de_nombres; $i++){
      echo $nombres[$i] . '<br />';
   }
?>
```

Esto imprime:

```
Juan
Pedro
Maria
```

Aquí la variable **$i**, que se va autoincrementando servirá para recorrer los sub índices del array.

Foreach

Aunque el anterior ejemplo del **for** para recorrer un array funcione, en PHP hay una alternativa mejor a la hora de recorrer este tipo de datos. El bucle foreach está pensado justamente para este tipo de cosas:

```
<?php
   $nombres = array('Juan', 'Pedro', 'Maria');
   foreach($nombres as $item){
      echo $item . '<br />';
   }
```

```
?>
```

Como se muestra en el ejemplo debemos indicarle al **foreach** por empezar el array que deberá recorrer seguido por la palabra reservada **as** y el alias que nos servirá para referirnos la posición en cada vuelta.

También podríamos recorrer un array asociativo de la misma forma:

```
<?php
   $perro = array(
      'Nombre' => 'Charly',
      'Color' => 'Blanco',
      'Raza' => 'Dogo'
   );
   foreach($perro as $item){
      echo $item . '<br />';
   }
?>
```

Lo que va a imprimir por pantalla:

```
Charly
Blanco
Dogo
```

Si lo que queremos es recuperar además de los valores los índices deberíamos hacerlo de la siguiente manera:

```
<?php
   $perro = array(
      'Nombre' => 'Charly',
      'Color' => 'Blanco',
      'Raza' => 'Dogo'
   );
   foreach($perro as $indice => $valor){
      echo "$indice: $valor <br />";
   }
?>
```

Y el resultado será:

```
Nombre: Charly
Color: Blanco
Raza: Dogo
```

También podemos recorrer un array multidimensional:

```
<?php
   $mercosur = array(
      array('nombre' => 'Argentina', 'moneda' => 'Peso argentino'),
      array('nombre' => 'Brasil', 'moneda' => 'Real'),
      array('nombre' => 'Paraguay', 'moneda' => 'Guaraní'),
      array('nombre' => 'Uruguay', 'moneda' => 'Peso uruguayo'),
      array('nombre' => 'Venezuela', 'moneda' => 'Bolívar fuerte'),
```

```
   );
   foreach ($mercosur as $item){
      echo "{$item['nombre']} / {$item['moneda']} <br />";
   }
?>
```

Esto devolverá:

```
Argentina / Peso argentino
Brasil / Real
Paraguay / Guaraní
Uruguay / Peso uruguayo
Venezuela / Bolívar fuerte
```

Break y continue

Break y **continue** no son bucles, pero pueden usarse dentro de estos.

En el caso de **break** romperá el ciclo de repetición omitiendo las vueltas que sigan. Por ejemplo vamos a tener un array con varios colores y lo recorreremos con un foreach, habrá una condición dentro de cada vuelta que buscará si la posición actual de la vuelta es el color verde, y de serlo finalizará el ciclo:

```
<?php
   $colores = array('Azul', 'Rojo', 'Amarillo', 'Verde', 'Naranja',
'Violeta');
   $color_buscado = 'Verde';
   foreach($colores as $item){
      echo $item . '... <br />';
      if($item == $color_buscado){
         echo '<strong> El color ha sido encontrado. Final. </strong>';
         break;
      }
   }
?>
```

Tenemos un array con distintos colores, como dijimos, recorremos la colección y buscamos el color ‘Verde’, de encontrarlo el ciclo de repetición finalizará. Por eso los últimos colores (‘Naranja’ y ‘Violeta’) no se muestran. Esto imprime por pantalla:

```
Azul...
Rojo...
Amarillo...
Verde...
El color ha sido encontrado. Final.
```

En el caso del **continue** no romperá el ciclo, pero salteará el resto de la vuelta. Por ejemplo, podríamos mostrar una lista desde del 1 al 10 y saltear aquellos números que sean múltiplos de 3 (3, 6 y 9)

```
<?php
   for($i=1; $i < 10; $i++){
      if($i % 3 == 0){
         continue;
      }
      echo $i . '<br />';
   }
?>
```

En cada bucle se mostrará un número del 1 al 10, pero antes se avaluará si el número que se va a imprimir es múltiplo de 3, y de serlo disparará un **continue** que hará que se ignore el resto de la vuelta. Entonces la salida al navegador será:

```
1
2
4
5
7
8
```

Funciones

En PHP existen algo llamado funciones, éstas son instrucciones que podemos darle a nuestro código, como si fueran comportamientos almacenados que al ser invocados hacen algo. Existes funciones declaradas por el usuario y funciones propias de PHP, en esta publicación veremos las primeras.

En primer lugar en PHP una función se crea mediante la palabra reservada **function** + el nombre de la misma + apertura y cierre de paréntesis y finalmente dentro de llaves las líneas de código que se ejecutarán al llamar a la función:

```
<?php
   function nombreDeLaFuncion(){
      //Lo que hace la función.
   }
?>
```

Por ejemplo podríamos crear una función:

```
<?php
   function saludar(){
      echo 'Hola!!!';
   }
?>
```

Esto no hará nada más que crear en memoria la función que puede ser invocada más adelante. Para invocarla debemos llamarla por su nombre + los paréntesis:

```
<?php
```

```
   function saludar(){
      echo 'Hola!!!';
   }
   saludar();
?>
```

Al ejecutar **saludar()** esto devolverá por pantalla:

```
Hola!!!
```

Parámetros

Los parámetros, también llamados argumentos, son datos que podemos pasar a nuestra función para realizar una operación. Por ejemplo podríamos tener una función **saludar()** como la anterior a la que habrá que pasar como parámetro un nombre, y saludará a ese nombre:

```
<?php
   function saludar($nombre){
      echo "Hola $nombre";
   }
   saludar('Fernando');
?>
```

Esto devolverá por pantalla:

```
Hola Fernando
```

Notar que los parámetros, cuando se crea una función, se escriben con el mismo formato que una variable con el signo **$** delante. Y además podemos agregar más de un parámetro:

```
<?php
   function sumar($num1, $num2){
      $resultado = $num1 + $num2;
      echo "El resultado de la suma es $resultado";
   }
   sumar(20, 7);
?>
```

Esto devolverá por pantalla:

```
El resultado de la suma es 27
```

Ahora bien, existen dos formas de pasar parámetros, por valor o por referencia. Hasta ahora lo hemos hecho por referencia, esto significa que las funciones no podrán alterar el código externo. Por ejemplo si tuviéramos algo cómo esto:

```
<?php
```

```
   function multiplicarPor2($num){
      $num *= 2;
   }
?>
```

A simple vista esta función recibe un número como parámetro y lo multiplica por 2, osea que si tuviéramos esto:

```
<?php
   function multiplicarPor2($num){
      $num *= 2;
   }
   $numero = 10;
   multiplicarPor2($numero);
   echo $numero;
?>
```

Podríamos deducir que el valor que devuelve por pantalla el script es 20, porque la función multiplica por 2 y 10 * 2 es igual a 20. Sin embargo el resultado de **$numero** sigue siendo:

```
10
```

Esto se debe, a como dije antes, las funciones que reciben datos como parámetros por valor no pueden modificar estos datos, ya que están por fuera de la función.

Sin embargo podríamos realizar esto mediante el pase de parámetros por referencia:

```
<?php
   function multiplicarPor2(&$num){
      $num *= 2;
   }
   $numero = 10;
   multiplicarPor2($numero);
   echo $numero;
?>
```

Para indicarle a nuestra función que un parámetro se pasa por referencia debemos usar el signo **&** delante del parámetro. En este caso, esto devolverá por pantalla:

```
20
```

Variables de entorno

Las variables que se declaran dentro de una función, nacen y mueren dentro de la misma, por tanto, si hiciéramos algo cómo esto:

```
<?php
   function setearVariable(){
      $nombre = 'Fernando';
   }
```

```
   setearVariable();
   echo $nombre;
?>
```

Esto nos devolverá un hermoso error:

```
Notice: Undefined variable: nombre
```

¿Y esto por qué? Porque cómo dijimos antes las variables que se declaran en una función, sólo existen dentro de la misma, si intentamos llamarla desde afuera el programa nos devolverá un error de tipo Notice (ya veremos los tipos de errores) porque para el programa esa variable no existe más que en la función. Por tanto, esto es muy importante de tener en cuenta de cómo trabaja una función, es como una especie de sub programa.

Los que sí podríamos hacer es crear dentro de la función una variable global, lo cual hará que está este disponible en el resto de la aplicación.

```
<?php
   function setearVariable(){
      global $nombre;
      $nombre = 'Fernando';
   }
   setearVariable();
   echo $nombre;
?>
```

En este caso la función **$nombre** sí funcionará en el resto del script a pesar de que se ha creado dentro de una función.

Variables estáticas

Como dijimos antes una variable declarada dentro de una función tiene utilidad dentro de la misma y al invocarse la función volverán a crearse todas las variables que estén dentro. Sin embargo a través de las variables estáticas PHP puede darnos la posibilidad de crear una función reutilizable en cada llamada:

```
<?php
   function aumentarNumero(){
      static $numero;
      $numero++;
      echo $numero . '<br />';
   }
   aumentarNumero();
   aumentarNumero();
   aumentarNumero();
?>
```

Esto nos devolverá por pantalla:

```
1
2
3
```

Retorno de una función

Dijimos que una función nos da la posibilidad de pasar datos de entradas, parámetros, pero las funciones también son capaces de devolvernos valores mediante la palabra reservada **return**. Por ejemplo podríamos crear una función llamada **sumar()** como la que creamos antes y devolver el resultado de la misma:

```
<?php
   function sumar($num1, $num2){
      $resultado = $num1 + $num2;
      return $resultado;
   }
   $resultado_de_suma = sumar(20, 7);
   echo $resultado_de_suma;
?>
```

Dentro de la variable **$resultado_de_suma** guardaremos lo que devuelve la función, así que al imprimir la variable por pantalla nos devolverá:

```
27
```

Una cosa que hay que tener en cuenta es que después de **return** el código que siga no será tenido en cuenta por el intérprete, por ejemplo:

```
<?php
   function sumar($num1, $num2){
      $resultado = $num1 + $num2;
      return $resultado;
      echo 'Esto no se tiene en cuenta';
   }
   $resultado_de_suma = sumar(20, 7);
   echo $resultado_de_suma;
?>
```

La línea:

```
echo 'Esto no se tiene en cuenta';
```

No se ejecutará debido a que está después de un **return**, y esto no sólo le indicará el valor que devuelve la función, sino que la misma ahí finaliza.

Además una función no puede devolver más de un valor, pero si en cambio puede devolver cualquier tipo de valor, por ejemplo un array:

```
<?php
   function sumar_restar($num1, $num2){
      $resultado = array(
         'suma' => $num1 + $num2,
         'resta' => $num1 - $num2
      );
```

```
      return $resultado;
   }
   $resultado_de_suma_resta = sumar_restar(20, 7);
   echo 'La suma es: ' . $resultado_de_suma_resta['suma'] . '<br />';
   echo 'La resta es: ' . $resultado_de_suma_resta['resta'] . '<br />';
?>
```

Los que nos devolverá por pantalla:

```
La suma es: 27
La resta es: 13
```

Funciones dentro de funciones

Dentro de una función podemos invocar otra sin ningún problema para así ahorrar líneas de código innecesaria. Por ejemplo podríamos tener dos funciones:

```
<?php
   function dividir($num1, $num2){
      $resultado = $num1 / $num2;
      return $resultado;
   }
   function verificarMultiplo2($num){
      if($num % 2 == 0){
         return true;
      }else{
         return false;
      }
   }
?>
```

La primer función divide dos números y retorna el resultado mientras que la segunda recibe un número y devuelve **true** si es múltiplo de 2 y **false** si no lo es:

```
<?php
   function dividir($num1, $num2){
      $resultado = $num1 / $num2;
      return $resultado;
   }
   function verificarMultiplo2($num){
      if($num % 2 == 0){
         return true;
      }else{
         return false;
      }
   }
   function dividir_y_verificar_multiplo_2($num1, $num2){
      $resultado = dividir($num1, $num2);
      $multiplo = verificarMultiplo2($resultado);
      return $multiplo;
   }
?>
```

La función **dividir_y_verificar_multiplo_2()** recibirá dos números y devolverá si el resultado de la división de ambos es múltiplo de 2, usando en su interior las otras dos funciones: **dividir()** y **verificarMultiplo2()**.

Funciones nativas

En el post pasado aprendimos para qué son las funciones y cómo crearlas. Pero en PHP ya existen en memoria funciones que nos permiten realizar distintas operaciones al invocarlas. A continuación veremos algunas de las funciones más utilizadas:

round()

Esta función nos permitirá hacer redondeo, osea recibirá como parámetro un número y si el valor después del punto (o de la coma) es menor a 50 redondeará para abajo, de lo contrario lo hará un número más:

```
<?php
 $decimal = 1.49;
 $decimal_redondeado = round($decimal);
 echo $decimal_redondeado;
?>
```

En este caso el resultado será:

```
1
```

Pero si en cambio modificamos el número decimal por **1.75**:

```
<?php
 $decimal = 1.75;
 $decimal_redondeado = round($decimal);
 echo $decimal_redondeado;
?>
```

Entonces el resultado será:

```
2
```

Esta función es muy útil a la hora de sacar promedios.

floor()

Esta función también hará un redondeo pero siempre para abajo, osea que si tenemos esto:

```
<?php
 $decimal = 1.50;
 $decimal_redondeado = floor($decimal);
 echo $decimal_redondeado;
?>
```

Nos devolverá por pantalla:

```
1
```

ceil()

Si en cambio queremos que el redondeo sea siempre para arriba deberíamos usar **ceil()**:

```
<?php
 $decimal = 1.50;
 $decimal_redondeado = ceil($decimal);
 echo $decimal_redondeado;
?>
```

En donde el resultado será por supuesto:

```
2
```

rand()

La función **rand()** es una de las más útiles que hay. Esta función generará un número al azar que irá desde un rango desde/hasta. Por ejemplo si quisiéramos devolver un número desde el 1 al 9 (inclusive):

```
<?php
 $numero_al_azar = rand(1, 9);
 echo $numero_al_azar;
?>
```

A mí me devolvió:

```
3
```

Pero puede devolver cualquiera de los número de los rangos comprendidos.

strtoupper()

Convierte una cadena en mayúscula;

```
<?php
 $palabra = 'Me encantan las arepas venezolanas.';
 $palabra_mayuscula = strtoupper($palabra);
 echo $palabra_mayuscula;
?>
```

Esto devolverá por pantalla:

```
ME ENCANTAN LAS AREPAS VENEZOLANAS.
```

strtolower()

Esta función es lo contrario a la anterior, convertirá una cadena pero en minúsculas:

```
<?php
 $palabra = 'No GRITES, habla bajito.';
 $palabra_minuscula = strtolower($palabra);
 echo $palabra_minuscula;
?>
```

Esto nos mostrará por pantalla:

```
no grites, habla bajito.
```

ucfirst()

En este caso, esta función sólo nos convertirá la primer letra de una cadena en mayúscula:

```
<?php
 $palabra = 'fernando';
 $nombre = ucfirst($palabra);
 echo $nombre;
?>
```

Devolverá por pantalla:

```
Fernando
```

ucwords()

En este caso, esta función recibirá una cadena y convertirá el mayúscula los primeros caracteres de cada palabra:

```
<?php
 $palabra = 'fernando gonzalo gaitan';
 $nombre_completo = ucwords($palabra);
 echo $nombre_completo;
?>
```

Esto va a imprimir:

```
Fernando Gonzalo Gaitan
```

strlen()

Nos dará la cantidad de caracteres que tiene una cadena:

```
<?php
 $cancion = 'De aquel amor de música ligera, nada nos libra, nada más queda';
 $cantidad_de_caracteres = strlen($cancion);
 echo $cantidad_de_caracteres;
?>
```

Esto da como resultado:

```
64
```

substr()

Esta función nos permitirá seleccionar cuántos caracteres queremos mostrar de una cadena. Recibe tres parámetros, el primero es el la cadena en sí, el segundo es a partir de qué caracter queremos que se muestre (empieza desde 0) y el tercero hasta cuál:

```
<?php
 $cancion = 'De aquel amor de música ligera, nada nos libra, nada más queda';
 $cancion_cortada = substr($cancion, 0, 11);
 echo $cancion_cortada;
?>
```

Esto nos dará como resultado:

```
De aquel am
```

trim()

Ésta es otra de las funciones más usadas de PHP, se trata de una función simple, pero muy útil, la misma cortará los espacios en las cadenas tanto adelante como atrás:

```
<?php
 $cadena = ' Hola ';
 $cadena_cortada = trim($cadena);
 echo $cadena_cortada;
?>
```

Para comprobar su correcto funcionamiento podemos utilizar la función **strlen()** y ver el comportamiento de **trim()**:

```
<?php
 $cadena = ' Hola ';
 $cadena_cortada = trim($cadena);
 echo strlen($cadena) . '<br />';
 echo strlen($cadena_cortada) . '<br />';
?>
```

Esto por pantalla devolverá:

```
6
4
```

Como vemos la segunda variable tendrá sólo 4 caracteres ya que los caracteres de adelante y atrás se eliminarán.

Sin embargo **trim()** no sólo funciona para eliminar espacios al principio y al final, puede eliminar cualquier tipo de caracter si le agregamos un segundo parámetro con el mismo:

```
<?php
 $cadena = '///Hola///';
 $cadena_cortada = trim($cadena, '/');
 echo $cadena_cortada;
?>
```

Al imprimir el resultado por pantalla veremos:

```
Hola
```

str_replace()

La función **str_replace()** nos permitirá buscar una cadena específica y reemplazarla por otra, por ejemplo podríamos reemplazar guiones bajo por guiones del medio:

```
<?php
 $frase = 'Marge_creo_que_odio_a_Mickael_Jackson.';
```

```
 $frase_modificada = str_replace('_', '-', $frase);
 echo $frase_modificada;
?>
```

El resultado será:

```
Marge-creo-que-odio-a-Mickael-Jackson.
```

Pero como dijimos antes, se pueden modificar cadenas, osea no sólo un caracter:

```
<?php
 $frase = 'Una manzana y una manzana son dos manzanas';
 $frase_modificada = str_replace('manzana', 'naranja', $frase);
 echo $frase_modificada;
?>
```

Esto imprime por pantalla:

```
Una naranja y una naranja son dos naranjas
```

var_dump()

La función **var_dump()** es muy útil en tiempo de desarrollo a la hora de testear datos ya sean de una variable o de lo que devuelve una función. Esta función nos devolverá la estructura de una variable cualquiera:

```
<?php
 $cadena = 'Manzana';
 $numero = 5;
 $valor_logico = true;
 $coleccion = array('Azul', 'Rojo', 'Verde');
 echo '<pre>';
 var_dump($cadena) . '<br />';
 var_dump($numero) . '<br />';
 var_dump($valor_logico) . '<br />';
 var_dump($coleccion) . '<br />';
 echo '</pre>';
?>
```

Esto devuelve:

```
string(7) "Manzana"
int(5)
bool(true)
array(3) {
  [0]=>
  string(4) "Azul"
  [1]=>
  string(4) "Rojo"
  [2]=>
  string(5) "Verde"
```

```
}
```

sort()

La función **sort()** permite ordenar un array alfabéticamente, vale aclarar que el valor del parámetro se hace por referencia, por tanto éste se modificará directamente:

```
<?php
 $nombres = array('Juan', 'Pedro', 'Maria', 'Jorge', 'Ana');
 sort($nombres);
 foreach($nombres as $item){
    echo $item . '<br />';
 }
?>
```

Esto imprimirá la lista ordenada de nombres:

```
Ana
Jorge
Juan
Maria
Pedro
```

rsort()

Funciona igual que **sort()**, sólo que el orden lo hace de forma descendente:

```
<?php
 $nombres = array('Juan', 'Pedro', 'Maria', 'Jorge', 'Ana');
 rsort($nombres);
 foreach($nombres as $item){
    echo $item . '<br />';
 }
?>
```

El resultado es:

```
Pedro
Maria
Juan
Jorge
Ana
```

shuffle()

Desordena las posiciones de un array. También se pasa por referencia:

```
<?php
 $nombres = array('Juan', 'Pedro', 'Maria', 'Jorge', 'Ana');
 shuffle($nombres);
 foreach($nombres as $item){
    echo $item . '<br />';
 }
?>
```

Por ejemplo en mi caso me ha devuelto:

```
Maria
Ana
Jorge
Juan
Pedro
```

array_push()

Agrega una posición nueva al final de un array, recibe dos parámetros, el mismo array y el valor nuevo:

```
<?php
 $nombres = array('Juan', 'Pedro', 'Maria', 'Jorge', 'Ana');
 array_push($nombres, 'Carlos');
 foreach($nombres as $item){
    echo $item . '<br />';
 }
?>
```

El resultado es:

```
Juan
Pedro
Maria
Jorge
Ana
Carlos
```

array_unshift()

A diferencia de **array_push()** agrega una nueva posición, pero al principio:

```
<?php
 $nombres = array('Juan', 'Pedro', 'Maria', 'Jorge', 'Ana');
 array_unshift($nombres, 'Carlos');
 foreach($nombres as $item){
```

```
    echo $item . '<br />';
 }
?>
```

En este caso el resultado será:

```
Carlos
Juan
Pedro
Maria
Jorge
Ana
```

in_array()

Buscará un valor dentro de un array, si lo encuentra devuelve **true** y sino **false**:

```
<?php
 $nombres = array('Juan', 'Pedro', 'Maria', 'Jorge', 'Ana');
 $nombre_a_buscar = 'Jorge';
 if(in_array($nombre_a_buscar, $nombres)){
    echo 'El nombre se encuentra en la lista';
 }else{
    echo 'El nombre NO se encuentra en la lista';
 }
?>
```

En nuestro caso como el nombre ‘Jorge’ está en el array devolverá:

```
El nombre se encuentra en la lista
```

count()

Devuelve la cantidad de posiciones que tiene un array:

```
<?php
 $nombres = array('Juan', 'Pedro', 'Maria', 'Jorge', 'Ana');
 $nombres_total = count($nombres);
 echo $nombres_total;
?>
```

El resultado es:

```
5
```

explode()

La función **explode()** convertirá un string en un array indicándole como referencia un valor que permitirá separar las distintas posiciones. Por ejemplo si tuviéramos una cadena:

```
$paises_cadena = 'Argentina - Bolivia - Brasil - Chile - Paraguay - Uruguay';
```

Y quisiéramos convertir esto en un array en donde cada posición será un país, deberíamos usar un delimitador, que en este caso es el signo '-':

```
<?php
 $paises_cadena = 'Argentina - Bolivia - Brasil - Chile - Paraguay - Uruguay';
 $paises_arreglo = explode('-', $paises_cadena);
 foreach($paises_arreglo as $item){
    echo $item . '<br />';
 }
?>
```

El resultado será entonces:

```
Argentina
Bolivia
Brasil
Chile
Paraguay
Uruguay
```

impode()

Esta función es lo inverso a **explode()**, convierte un array en un string, y concatenará la cadena con las distintas posiciones por un delimitador:

```
<?php
 $paises_arreglo = array('Colombia', 'Ecuador', 'Peru', 'Venezuela');
 $paises_cadena = implode(' - ', $paises_arreglo);
 echo $paises_cadena;
?>
```

El resultado será:

```
Colombia - Ecuador - Peru - Venezuela
```

En PHP existen muchísimas funciones, las que vimos son sólo algunas, pero hay muchas más. Cuando necesitamos realizar alguna operación es cuestión de buscar en el manual de PHP:

https://php.net/

Saludos!

Fechas

En PHP se puede trabajar con fechas al igual que cualquier otro lenguaje de programación. Esto es muy útil para tener un control en nuestra aplicación web del tiempo.

date()

La función **date()** nos permite formatear una fecha, ya sea con el día, mes, año, horas, minutos, segundos, día de la semana, etc. Ésta recibirá dos parámetros, el primero será justamente el formato de la fecha que queremos mostrar, y un segundo parámetro opcional que es el **timestamp** (ya veremos lo qué es eso)

Por ejemplo si quisiéramos devolver el año, mes y día:

```
<?php
 $fecha = date('Y-m-d');
 echo $fecha;
?>
```

Esto nos devolverá por pantalla la fecha en formato en inglés (2013 septiembre 19):

```
2013-09-19
```

'Y' hace referencia al año (Year en inglés), 'm' al mes (month) y 'd' al día (day) Pero también podemos formatear el mismo en español:

```
<?php
 $fecha = date('d-m-Y');
 echo $fecha;
?>
```

Entonces el resultado será:

```
19-09-2013
```

También podemos recuperar la fecha y la hora:

```
<?php
 $fecha_hora = date('d-m-Y H:i:s');
 echo $fecha_hora;
?>
```

En mi caso, en el momento es que estoy haciendo esto, el resultado que me dio es:

```
19-09-2013 19:32:26
```

mktime()

Otra forma de recuperar la fecha y la hora en PHP es en formato timestamp (o marca de tiempo Unix), esto es la cantidad de segundos que va desde de la época Unix (1 de Enero de 1970 a las 00:00:00) hasta el momento en que se llama. Esta función recibirá 6 parámetros: la hora, los minutos, los segundos, el mes, el día y el año (en ese orden):

```
<?php
 $mktime = mktime('19', '50', '02', '09', '19', '2013');
 echo $mktime;
?>
```

Esto nos devolverá por pantalla el valor:

```
1379631002
```

Lo que corresponde al 19 de Septiembre del 2013 a las 19:50:02.

Por ejemplo si quisiéramos recuperar la cantidad de segundos que hay desde esa fecha hasta el 19 de Septiembre del 2013 a las 18:50:02:

```
<?php
 $mktime1 = mktime('19', '50', '02', '09', '19', '2013');
 $mktime2 = mktime('18', '50', '02', '09', '19', '2013');
 $resta = $mktime1 - $mktime2;
 echo $resta;
?>
```

Esto nos devolverá por pantalla:

```
3600
```

Que es la diferencia de segundos que hay desde una hora a la otra.

Ahora si quisiéramos devolver la fecha y la hora a través de un valor timestamp deberíamos hacer lo siguiente:

```
<?php
 $fecha = date('d-m-Y H:i:s', 1379631002);
 echo $fecha;
?>
```

Como se ve, esta vez la función **date()** recibirá un segundo parámetro que es el que le pasará la fecha que deberá formatear.

El resultado será:

```
19-09-2013 19:50:02
```

time()

Otra función que nos permitirá devolver un valor de tipo timestamp es **time()**, en este caso recuperaremos el valor del momento en que la estamos llamando:

```
<?php
 $tiempo = time();
 $fecha = date('d-m-Y H:i:s');
 echo $tiempo;
 echo '<br />';
 echo $fecha;
?>
```

Esto nos devolverá el timestamp y el formato de la fecha, en mi caso:

```
1379632173
19-09-2013 20:09:33
```

strtotime()

Hasta ahora hemos pasado fechas timestamp en formato de fecha normal, pero si lo que queremos es hacer la inversa, deberíamos usar la función **strtotime()** que nos devolverá el valor timestamp:

```
<?php
 $fecha = date('19-09-2013 20:09:33');
 $timestamp = strtotime($fecha);
 echo $timestamp;
?>
```

Esto devolverá por pantalla:

```
1379632173
```

La función **date()** no sólo puede devolver la fecha o la hora, para ver los otros tipos de formato disponibles podemos consultar el manual de PHP:

https://php.net/manual/es/function.date.php

Zona horaria

Hasta acá todo muy bonito, cada vez que pedimos una fecha o una hora PHP nos la devuelve, pero, ¿cuál es la hora qué toma? Es decir, cada país, o mejor dicho región, tiene distintos horarios. No es lo mismo la hora actual de Buenos Aires que la de México o Madrid. Si bien nuestro apache probablemente haya tomado la zona de nuestro país si lo probamos en el localhost, si decidimos subir nuestro proyecto a un hosting en donde sus servidores están en Estados Unidos, se tomará la zona horaria de esa región.

Por empezar para ver nuestra zona horaria actual debemos usar la función **date_default_timezone_get()**:

```
<?php
 echo date_default_timezone_get();
?>
```

En mi caso me ha devuelto:

```
America/Argentina/Buenos_Aires
```

Para modificar la zona horaria tendremos que usar la función **date_default_timezone_set()**, la misma recibirá un parámetro con la zona horaria. Por ejemplo para recuperar la fecha de Argentina:

```
<?php
 date_default_timezone_set('America/Argentina/Buenos_Aires');
 $fecha_argentina = date('H:i:s');
 echo $fecha_argentina;
?>
```

Esto me devolverá la hora de Argentina Buenos Aires:

```
21:12:47
```

Si en cambio quiero ver la hora de Guatemala, por ejemplo, debería hacer:

```
<?php
 date_default_timezone_set('America/Guatemala');
 $fecha_guatemala = date('H:i:s');
 echo $fecha_guatemala;
?>
```

Esto me devolverá obviamente otra hora:

```
18:13:24
```

Para ver el listado de zonas horarios podemos visitar la página de PHP:

https://www.php.net/manual/es/timezones.php

checkdate()

Esta función nos mete en un terreno que aun no hemos entrado, las validaciones. Esta función valida que una fecha sea correcta, recibe tres parámetros: mes, día y año, en ese orden. Si la fecha es correcta devolverá **true**, sino **false**:

```
<?php
 $dia = 31;
 $mes = 11;
 $anio = 1986;
 if(checkdate($mes, $dia, $anio)){
    echo 'La fecha es correcta';
 }else{
    echo 'La fecha es incorrecta';
 }
?>
```

En este caso la función nos devolverá:

```
La fecha es incorrecta
```

Porque obviamente 31 de Noviembre de 1986 no es una fecha correcta, el mes de Noviembre tiene hasta el día 30.

Trabajar con archivos

Php nos da la posibilidad de crear, modificar, eliminar y recuperar el contenido de archivos de texto como por ejemplo txt, xml o incluso html.

Crear archivos

Para crear y editar archivos debemos usar dos funciones, **fopen()** y **fwrite()**, la primer función creará un archivo si no existe, y si existe lo reemplazará por el nuevo. **fwrite()** nos permitirá escribir el contenido del mismo:

```
<?php
 $contenido = 'Texto de prueba';
 $archivo = fopen('archivo.txt', 'w');
 fwrite($archivo, $contenido);
 fclose($archivo);
?>
```

Como se ve en el ejemplo creamos un texto para el archivo: ”Texto de prueba’, luego abrimos el archivo al que llamamos ‘archivo.txt’, con respecto al segundo parámetro se refiere al la manera de abrir el mismo, en nuestro caso es w de ‘write’, osea de escritura. Los modos son:

- **w** para escritura, si el archivo no existe lo crea y si ya existe elimina sus contenidos.
- **w+** para escritura y lectura, si el archivo no existe lo crea y si ya existe elimina sus contenidos.
- **r** para lectura
- **r+** para lectura y escritura
- **a** para adjuntar, escribe al final del mismo, si no existe intenta crearlo.
- **b** para archivos binarios, este es el recomendado para php.

Luego usamos la función **fwrite()** para escribir el contenido del archivo y finalmente cerramos el mismo con **fclose()**.

Leer archivos

Para leer un archivo debemos usar la función **file_get_contents()** que nos devolverá todo el texto de éste. Por ejemplo si quisiéramos recuperar el archivo que acabamos de crear:

```
<?php
 $contenido_de_archivo = file_get_contents('archivo.txt');
 echo $contenido_de_archivo;
?>
```

Esto nos devolverá:

```
Texto de prueba
```

Que obviamente es el texto que le agregamos al archivo.

Buscar archivos

Hasta acá todo muy bien, pero nosotros no hemos tenido en cuenta algo, ¿qué pasa si el archivo no existe? Si por ejemplo nosotros quisiéramos leer un archivo que no existe PHP nos devolverá un error. Por tanto antes de leer el archivo debemos comprobar que éste exista con la función **file_exists()**:

```
<?php
 if(file_exists('archivo.txt')){
    echo '<strong> El contenido del archivo es: </strong>';
    echo '<br />';
    echo file_get_contents('archivo.txt');
 }else{
    echo '<strong> El archivo no existe </strong>';
 }
?>
```

Eliminar archivos

Para eliminar archivos debemos usar la función **unlink()** que eliminará el mismo. Devolverá true o false dependiendo de si se pudo o no eliminar el archivo:

```
<?php
 if(file_exists('archivo.txt')){
    if(unlink('archivo.txt')){
       echo '<strong> El archivo ha sido eliminado </strong>';
    }else{
       echo '<strong> Error al intentar eliminar el archivo </strong>';
    }
 }else{
    echo '<strong> El archivo no existe </strong>';
 }
?>
```

Crear directorios

Para crear un directorio debemos usar la función **mkdir()** que recibirá como parámetro el nombre del directorio que vamos a crear:

```
<?php
 if(mkdir('carpeta')){
    echo 'El directorio ha sido creado';
 }else{
    echo 'El directorio NO ha podido ser creado';
 }
?>
```

Eliminar directorios

Para eliminar un directorio debemos comprobar si el mismo existe con la función **is_dir()** y eliminarlo con **rmdir()**, ambos recibirán como parámetro el nombre del directorio:

```
<?php
 if(is_dir('carpeta')){
    if(rmdir('carpeta')){
       echo 'El directorio ha sido eliminado';
    }else{
       echo 'El directorio no ha podido ser eliminado';
    }
 }else{
    echo 'El directorio no existe';
 }
?>
```

Listar archivos de un directorio

Con PHP podemos recorrer los archivos de un directorio en forma de array. Supongamos que creamos un directorio con tres archivos:

```
<?php
 mkdir('animales');
 $perro = fopen('animales/perro.txt', 'w');
 $caballo = fopen('animales/caballo.txt', 'w');
 $tigre = fopen('animales/tigre.txt', 'w');
 fclose($perro);
 fclose($caballo);
 fclose($tigre);
?>
```

Y ahora debemos usar la función **scandir()** que nos devolverá el listado con los nombres de archivos. Recibirá como parámetro el nombre del directorio:

```
<?php
 $animales = scandir('animales');
 foreach($animales as $archivo){
    echo $archivo . '<br />';
 }
?>
```

Que devolverá por pantalla:

```
.
..
caballo.txt
perro.txt
tigre.txt
```

Notar que también recupera el punto(.) y doble punto punto(..), para quitarlos simplemente debemos filtrarlos con un **if**:

```
<?php
 $animales = scandir('animales');
 foreach($animales as $archivo){
    if($archivo != '.' and $archivo != '..'){
       echo $archivo . '<br />';
    }
 }
?>
```

Copiar archivos

Supongamos que nosotros quisiéramos copiar un archivo de un directorio y pegarlo en otro, deberíamos usar la función **copy()** que recibirá dos parámetros, la ruta donde está el archivo actualmente y dónde queremos copiar:

```
<?php
 //Creamos un directorio
 mkdir('directorio1');
 //Creamos un archivo dentro del directorio que acabamos de crear
 $archivo = fopen('directorio1/archivo.txt', 'w');
 fclose($archivo);
 //Creamos un segundo directorio
 mkdir('directorio2');
 //Copiamos el archivo en el nuevo directorio
 copy('directorio1/archivo.txt', 'directorio2/archivo.txt');
?>
```

Pero si lo que en cambio queremos hacer es mover un archivo de un directorio a otro deberíamos usar la función **rename()**, que recibirá los mismos parámetros que **copy()**:

```
<?php
 //Creamos un directorio
 mkdir('directorio1');
 //Creamos un archivo dentro del directorio que acabamos de crear
 $archivo = fopen('directorio1/archivo.txt', 'w');
 fclose($archivo);
 //Creamos un segundo directorio
 mkdir('directorio2');
 //Movemos el archivo en el nuevo directorio
 rename('directorio1/archivo.txt', 'directorio2/archivo.txt');
?>
```

Para entender mejor la diferencia, **copy()** es copy/paste (copiar y pegar), mientras que **rename()** es cortar/pegar (cortar y pegar) Además algo que no tuvimos en cuenta, pero es importarte agregar es que ambas funciones devuelve true o false, dependiendo de su pudieron o no realizar el propósito.

Información del archivo

Supongamos que tenemos un archivo creado:

```
<?php
 $archivo = fopen('archivo.txt', 'w');
 fwrite($archivo, 'Texto del archivo');
 fclose($archivo);
?>
```

Para recuperar la información del mismo debemos usar la función pathinfo():

```
<?php
 $info_de_archivo = pathinfo('archivo.txt');
```

```
 echo '<pre>';
 var_dump($info_de_archivo);
 echo '</pre>';
?>
```

Esto nos devolverá un array asociativo con información del archivo:

```
array(4) {
  ["dirname"]=>
  string(1) "."
  ["basename"]=>
  string(11) "archivo.txt"
  ["extension"]=>
  string(3) "txt"
  ["filename"]=>
  string(7) "archivo"
}
```

También podemos recuperar el peso del archivo en bytes:

```
<?php
 $peso_de_archivo = filesize('archivo.txt');
 echo $peso_de_archivo;
?>
```

Errores

PHP como cualquier otro lenguaje de programación puede generar errores. Algo importante de repasar es que existen lenguajes de programación interpretados y compilados. Un lenguaje de programación compilado como Java, en donde el programador escribe su código y antes de probar cómo está quedando su aplicación un compilador le indicará si hay errores, y de haberlo se frenará todo y el compilador le indicará al programador el problema.

Pero PHP no funciona de la misma forma, PHP es un lenguaje de programación interpretado, esto significa que nosotros vamos a escribir el código y si tenemos algún error el programa igual se ejecutará, luego el intérprete, en nuestro caso Apache, encontrará el error y detendrá su ejecución dependiendo de la gravedad del error.

Ahora bien, los errores no sólo pueden darse por errores del programador, por ejemplo por escribir con la sintaxis de forma incorrecta; también pueden darse factores que son ajenos al programador, por ejemplo si nosotros escribimos el código correcto para conectarse con una base de datos y de pronto alguien apaga la base de datos, nuestra aplicación no podrá conectarse y PHP lanzará un error por pantalla.

De todas formas, y desde mi punto de vista, tener o no errores no nos hace un mejor o peor programador, pero tener la capacidad y voluntad para evitarlos o resolverlos sí.

Errores más comunes

Existen varios errores que pueden darse en nuestro programa por errores del código, a continuación se detallan los más comunes:

Parse error

Estos errores se dan por un error de sintaxis en el código, por ejemplo si tenemos algo como esto:

```
function saraza{

}
```

Esto devolverá por pantalla:

```
Parse error: syntax error, unexpected '{', expecting '(' in
```

Por ejemplo, en este caso creamos una función y nos olvidamos de ponerle los paréntesis, por tanto el compilador encontrará el error y se romperá el script. Esto significa que todo lo que esté por debajo el intérprete va a ignorarlo.

Notice

Los errores de tipo **Notice**, son los errores 'menos graves' por así decirlo, pero esto no significa que no haya que corregirlos. Por ejemplo imprimir una variable que no está definida:

```
<?php
 echo $variable_que_no_existe;
?>
```

Esto imprime por pantalla:

```
Notice: Undefined variable: variable_que_no_existe in
```

Estos errores serán tenidos en cuenta por el intérprete, pero éste continuará con su ejecución, por ejemplo si tuviéramos algo como esto:

```
<?php
 echo $variable_que_no_existe;
 $variable_que_existe = 15;
 echo $variable_que_existe;
?>
```

En este caso se imprimirá por pantalla:

```
Notice: Undefined variable: variable_que_no_existe in
15
```

Osea, el intérprete nos informó del error, pero continuó con su ejecución, de hecho declaramos una variable con el valor 15 y lo imprimimos por pantalla.

En realidad cuando intentamos imprimir una variable que no está definida, si bien el intérprete lo toma como error inmediatamente le pasa como valor **null**.

Muchas veces los errores de tipo **Notice** suelen ser deshabilitados para que el intérprete no los imprima por pantalla, pero esto no es correcto. Por ejemplo si tuviéramos algo como esto:

```
<?php
 $variable_que_existe = 15;
 $resultado = $variable_que_no_existe + $variable_que_existe;
 echo $resultado;
?>
```

En este caso como **$variable_que_no_existe** no está definida, y cuando sumamos **$variable_que_existe**, que vale 15 con **$variable_que_no_existe**, el intérprete le asignará un valor 0:

```
Notice: Undefined variable: variable_que_no_existe in
```

Warning

Los errores de tipo **warning** son errores que, al igual que los de tipo **Notice**, no romperán nuestro script, sin embargo nos están advirtiendo de que es muy probable que en las siguientes líneas algo no funcione correctamente debido al error que se está generando en la actual. Por ejemplo, si quisiéramos recuperar el contenido de un archivo que no existe:

```
<?php
 $contenido = file_get_contents('archivo_que_no_existe.txt');
?>
```

Nos devolverá por pantalla:

```
Warning: file_get_contents(archivo_que_no_existe.txt): failed to open
stream: No such file or directory in
```

Esto puede perjudicar nuestra aplicación más adelante, si por ejemplo luego intentamos imprimir el contenido del archivo (que no existe):

```
<?php
 $contenido = file_get_contents('archivo_que_no_existe.txt');
 echo 'El contenido del archivo es: ' . $contenido;
?>
```

Esto nos devolverá:

```
Warning: file_get_contents(archivo_que_no_existe.txt): failed to open
stream: No such file or directory in
El contenido del archivo es:
```

Como se ve en el ejemplo, después del **Warning** intenta mostrar el contenido del archivo, pero como el archivo no existe entonces imprime una cadena vacía después de 'El contenido del archivo es:'

Fatal error

Los tipos de errores **Fatal error** romperán nuestro script a partir de la línea donde se producen, ya que el compilador no sabrá cómo continuar con la ejecución e ignorará las líneas que siguen:

```
<?php
 $resultado = sumar(10, 5);
?>
```

Esto devuelve:

```
Fatal error: Call to undefined function sumar() in
```

En este caso estamos llamando a una función que nunca ha sido definida llamada **sumar()**, el compilador no encuentra esta función y rompe.

Reportar errores

Ahora bien, este tipo de errores pueden mostrarse o no en nuestro script mediante la función **error_reporting()**, la misma recibirá un parámetro con los errores que queremos que se muestren:

Los valores posibles son:

- E_ERROR
- E_WARNING
- E_PARSE
- E_NOTICE

También pueden ser combinados mediante el signo |:

```
error_reporting(E_ERROR | E_WARNING | E_PARSE);
```

Si queremos reportar todo tipo de errores debemos usar **E_ALL**:

```
error_reporting(E_ALL);
```

Y para no reportar ninguno:

```
error_reporting(0);
```

Por ejemplo en mi caso, cuando yo estoy creando una aplicación uso **E_ALL**, ya que nosotros deberíamos tener un control absoluto de los errores en etapa de desarrollo, y al subirlos a la web cambio ese valor por **0**, porque una vez productivo nuestro sitio no debería mostrar errores de haberlos.

Manejo de errores

Hasta ahora nosotros vimos cómo a través de un error PHP nos muestra un mensaje por pantalla, pero nosotros también podríamos controlar esto:

Ignorar errores

Mediante el caracter **@** nosotros podemos ignorar líneas de código que pudiesen llegar a devolver errores. Por ejemplo:

```
<?php
 $archivo = @file_get_contents('archivo_que_no_existe.txt');
 if($archivo){
    echo $archivo;
 }else{
    echo 'El archivo no ha podido ser encontrado';
 }
?>
```

Como se ve en el ejemplo al llamar a la función **file_get_contents()** y como parámetro un archivo que no existe, entonces evitamos el error que devuelve en ese punto ya que el archivo no existe:

```
@file_get_contents('archivo_que_no_existe.txt');
```

También nosotros podemos personalizar nuestros errores con excepciones. Por ejemplo podríamos crear una función para leer archivos y luego utilizar las sentencias **try catch** para ejecutarlo y hacer algo se produce un error:

```
<?php
 function leerArchivo($archivo){
    if(file_exists($archivo)){
       return file_get_contents($archivo);
    }else{
       throw new Exception('El archivo no se puede leer porque no
existe.');
       return false;
    }
 }
```

```
 try{
    $texto = leerArchivo('archivo_que_no_existe.txt');
    echo $texto;
 }catch(PDOException $e){
    echo $e->getMessage();
    exit;
 }
?>
```

Por empezar creamos una función que intentará leer un archivo, de encontrarlo devolverá su contenido, pero sino devolverá **false** y creará una excepción con el mensaje 'El archivo no se puede leer porque no existe.' Esto de las excepciones lo veremos mucho más adelante ya que está relacionado con PHP orientado a objetos. Luego a través de la sentecia **try** intentamos recuperar el contenido del archivo y mostrarlo por pantalla, en el caso de **catch**, esto se ejecutará si encuentra un error dentro de **try**, en nuestro caso si el archivo no existe.

Gestor de errores

En el manual de PHP nosotros podremos encontrar una función para tratar los errores a nuestro gusto:

https://php.net/manual/es/function.set-error-handler.php

Por ejemplo:

```
<?php
 function miGestorDeErrores($errno, $errstr, $errfile, $errline) {
    if (!(error_reporting() & $errno)) {
       // Este código de error no está incluido en error_reporting
       return;
    }
    switch ($errno) {
       case E_USER_ERROR:
          echo "<b>Mi ERROR</b> [$errno] $errstr<br />\n";
          echo  "  Error  fatal  en  la  línea  $errline  en  el  archivo
$errfile";
          echo ", PHP " . PHP_VERSION . " (" . PHP_OS . ")<br />\n";
          echo "Abortando...<br />\n";
          exit(1);
          break;
       case E_USER_WARNING:
          echo "<b>Mi WARNING</b> [$errno] $errstr<br />\n";
          break;
       case E_USER_NOTICE:
          echo "<b>Mi NOTICE</b> [$errno] $errstr<br />\n";
          break;
       default:
          echo "Tipo de error desconocido: [$errno] $errstr<br />\n";
          break;
    }
    /* No ejecutar el gestor de errores interno de PHP */
```

```
    return true;
 }
 // establecer el gestro de errores definido por el usuario
 $gestor_errores_antiguo = set_error_handler("miGestorDeErrores");
 //Generamos un error imprimiendo una variable que no existe.
 echo $variable_que_no_existe
?>
```

Dentro del bloque:

```
switch ($errno) {
       case E_USER_ERROR:
          echo "<b>Mi ERROR</b> [$errno] $errstr<br />\n";
          echo  "  Error  fatal  en  la  línea  $errline  en  el  archivo
$errfile";
          echo ", PHP " . PHP_VERSION . " (" . PHP_OS . ")<br />\n";
          echo "Abortando...<br />\n";
          exit(1);
          break;
       case E_USER_WARNING:
          echo "<b>Mi WARNING</b> [$errno] $errstr<br />\n";
          break;
       case E_USER_NOTICE:
          echo "<b>Mi NOTICE</b> [$errno] $errstr<br />\n";
          break;
       default:
          echo "Tipo de error desconocido: [$errno] $errstr<br />\n";
          break;
}
```

Nosotros podemos cambiar el código por lo que queremos que suceda cuando se genera un error. Por ejemplo crear un archivo con el error.

Importar archivos

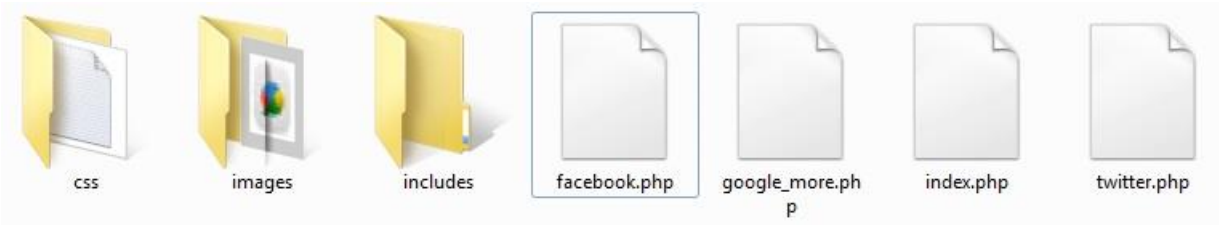
En un comienzo los sitios web se resumían a unas pocas páginas estáticas desarrolladas prácticamente sólo con código Html, algunas ni siquiera tenían código Javascript y Css tampoco existía en los primeros años. Estas pocas páginas solían repetir código Html, pero esto no era demasiado problema, ya que los cambios debían ajustarse en unas 4 ó 5 páginas, y aunque no dejaba de ser redundante no era un trabajo tampoco de esclavo.

Con el tiempo y con el avance de la web los sitios pasaron a ser de simples archivos .html con imágenes a aplicaciones tan complejas como las aplicaciones de escritorio, esto llevo a buscar lenguajes de servidor como Perl, Asp, Java o PHP. El problema es que ahora las webs ya no eran estáticas, sino dinámicas y su mantenimiento empezó a requerir nuevas soluciones.

Ahora viene lo que debemos aprender cada día como programadores, lo que nos hace más prácticos a la hora de trabajar: la reutilización, esto no es fácil, pero al menos hay que

intentarlo. Porque, y aunque parezca contradictorio, el programador más experimentado es aquel que cada vez escribe menos código.

Comencemos, para eso vamos a crear un nuevo proyecto en la carpeta htdocs (si instalaste Xampp) al que llamaremos **ejemplo_importar_archivos**, y dentro de éste vamos a crear cuatro archivos .php: index.php, facebook.php, twitter.php y google_more.php. También crearemos una carpeta llamada **images** y otra llamada **css**. Y por último una última llamada **includes** que de momento quedará vacía.



Y dentro del de la carpeta **images** vamos a copiar las siguientes imágenes (seis en total):

Y ahora vamos a entrar a la carpeta **css** y dentro vamos a crear un archivo llamado **style.css** con el siguiente código:

```
body {
   background-color: #D4ECF8;
}
#contenedor {
   background-color: #fff;
   width: 900px;
   margin: 0px auto 0px auto;
   padding: 10px;
   border: solid 1px #ccc;
   box-shadow: 1px 1px 5px #000;
   -moz-box-shadow: 1px 1px 5px #000;
   -webkit-box-shadow: 1px 1px 5px #000;
}
header {
   text-align: center;
}
nav ul{
   width: 450px;
   margin: 0px auto 0px auto;
}
nav li{
   float: left;
   width: 120px;
   height: 20px;
   list-style: none;
   border: solid 1px #999;
   margin-right: 10px;
   background-repeat: no-repeat;
   -moz-border-radius: 5px 15px 5px 5px;
   -webkit-border-radius: 5px 15px 5px 5px;
   text-align: center;
}
nav .google_mas {
   background-image: url(../images/google_mas_icono.jpg);
}
nav .facebook {
   background-image: url(../images/facebook_icono.jpg);
}
nav .twitter {
   background-image: url(../images/twitter_icono.jpg);
}
nav a {
   font-weight: bold;
   color: #000;
   text-decoration: none;
}
section .imagen_logo {
   float: left;
   width: auto;
}
section .descipcion p {
```

```
   text-align: justify;
}
.clear {
   clear: both;
}
```

Ok y ahora vamos a editar nuestro archivo index.php con el siguiente código:

```
<!DOCTYPE>
<html>
 <head>
    <title> Ejemplo importar archivos </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link type="text/css" href="css/style.css" rel="stylesheet" />
 </head>
 <body>
    <div id="contenedor">
       <header>
          <h1> Redes sociales más populares </h1>
       </header>
       <nav>
          <ul>
             <li  class="google_mas">  <a  href="google_more.php">  Google+
</a> </li>
             <li  class="facebook"> <a href="facebook.php"> Facebook </a>
</li>
             <li  class="twitter">  <a  href="twitter.php">  Twitter  </a>
</li>
          </ul>
          <div class="clear"></div>
       </nav>
       <section>
          <div class="descripcion">
             <h2> Redes sociales </h2>
             <p>  Una  red  social  es  una  forma  de  representar  una
estructura social, asignándole un grafo, si dos elementos del conjunto de
actores (tales como individuos u organizaciones) están relacionados de
acuerdo a algún criterio (relación profesional, amistad, parentesco,
etc.)  entonces  se  construye  una  línea  que  conecta  los  nodos  que
representan a dichos elementos. El tipo de conexión representable en una
red social es una relación diádica o lazo interpersonal, que se pueden
interpretar como relaciones de amistad, parentesco, laborales, entre
otros. </p>
             <p>       Extraído       de:       <a       target="_blank"
href="https://es.wikipedia.org/wiki/Redes_sociales"> Wikipedia </a> </p>
          </div>
          <div class="clear"></div>
       </section>
       <hr />
       <footer> <p> Ejemplo de importar archivos en PHP </p> </footer>
     </div>
 </body>
</html>
```

Bien, nada del otro mundo un simple código html con tres enlaces a otras páginas. Así que ahora seguimos, vamos a editar el código de las mismas:

google_more.php:

```
<!DOCTYPE>
<html>
 <head>
    <title> Ejemplo importar archivos </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link type="text/css" href="css/style.css" rel="stylesheet" />
 </head>
 <body>
    <div id="contenedor">
       <header>
          <h1> Redes sociales más populares </h1>
       </header>
       <nav>
          <ul>
             <li  class="google_mas"> <a  href="google_more.php"> Google+
</a> </li>
             <li  class="facebook"> <a  href="facebook.php"> Facebook </a>
</li>
             <li  class="twitter"> <a  href="twitter.php"> Twitter  </a>
</li>
          </ul>
          <div class="clear"></div>
       </nav>
       <section>
          <div class="imagen_logo">
             <img src="images/google_mas_logo.jpg" alt="Google" />
          </div>
          <div class="descripcion">
             <h2> Google+ </h2>
             <p> Google+ (pronunciado y a veces escrito Google Plus, a
veces abreviado como G+, en algunos países de lengua hispana pronunciado
Google Más) es un servicio de red social operado por Google Inc. El
servicio, lanzado el 28 de junio de 2011, está basado en HTML5. Los
usuarios tienen que ser mayores de 18 años de edad,2 para crear sus
propias cuentas. Google+ ya es la segunda red social más popular del
mundo con aproximadamente 343 millones de usuarios activos. </p>
             <p> Google+ integra los servicios sociales, tales como
Google Perfiles y Google Buzz, e introduce los nuevos servicios:
Círculos, Quedadas, Intereses y Mensajes.3 Google+ también estará
disponible como una aplicación de escritorio y como una aplicación móvil,
pero sólo en los sistemas operativos Android e iOS. Fuentes tales como
The New York Times lo han declarado el mayor intento de Google para
competir con la red social Facebook,4 la cual tenía más de 750 millones
de usuarios en 2011.5 El 20 de septiembre de 2011, Google permitió la
creación de cuentas a usuarios con más de 18 años, con mejoras en sus
extensiones de videoconferencias.6 </p>
             <p>       Extraído       de:       <a       target="_blank"
href="https://es.wikipedia.org/wiki/Google%2B"> Wikipedia </a> </p>
          </div>
          <div class="clear"></div>
       </section>
       <hr />
       <footer> <p> Ejemplo de importar archivos en PHP </p> </footer>
    </div>
 </body>
```

```
</html>
```

facebook.php:

```
<!DOCTYPE>
<html>
 <head>
    <title> Ejemplo importar archivos </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link type="text/css" href="css/style.css" rel="stylesheet" />
 </head>
 <body>
    <div id="contenedor">
       <header>
          <h1> Redes sociales más populares </h1>
       </header>
       <nav>
          <ul>
             <li  class="google_mas"> <a  href="google_more.php"> Google+
</a> </li>
             <li class="facebook"> <a href="facebook.php"> Facebook </a>
</li>
             <li  class="twitter"> <a  href="twitter.php">  Twitter  </a>
</li>
          </ul>
          <div class="clear"></div>
       </nav>
       <section>
          <div class="imagen_logo">
             <img src="images/facebook_logo.jpg" alt="Facebook" />
          </div>
          <div class="descripcion">
             <h2> Facebook </h2>
             <p> Facebook (NASDAQ: FB) es una empresa creada por Mark
Zuckerberg y fundada junto a Eduardo Saverin, Chris Hughes y Dustin
Moskovitz consistente en un sitio web de redes sociales. Originalmente
era un sitio para estudiantes de la Universidad de Harvard, pero
actualmente está abierto a cualquier persona que tenga una cuenta de
correo electrónico. Los usuarios pueden participar en una o más redes
sociales, en relación con su situación académica, su lugar de trabajo o
región geográfica. </p>
             <p> Ha recibido mucha atención en la blogosfera y en los
medios de comunicación al convertirse en una plataforma sobre la que
terceros pueden desarrollar aplicaciones y hacer negocio a partir de la
red social. </p>
             <p> A mediados de 2007 lanzó las versiones en francés,
alemán y español traducidas por usuarios de manera no remunerada,6
principalmente para impulsar su expansión fuera de Estados Unidos, ya que
sus usuarios se concentran en Estados Unidos, Canadá y Reino Unido.
Facebook cuenta con más de 900 millones de miembros, y traducciones a 70
idiomas.7 8 En octubre de 2012, Facebook llegó a los 1,000 millones de
usuarios, de los cuáles hay más de 600 millones de usuarios móviles.
Brasil, India, Indonesia, México y Estados Unidos son los países con el
mayor número de usuarios.9 </p>
             <p> Su infraestructura principal está formada por una red de
más de 50 000 servidores que usan distribuciones del sistema operativo
GNU/Linux usando LAMP.10 </p>
```

```
            <p> El 9 de abril de 2012, se anunció que Facebook adquirió
Instagram por mil millones de dólares.11 </p>
            <p>       Extraído       de:       <a       target="_blank"
href="https://es.wikipedia.org/wiki/Facebook"> Wikipedia </a> </p>
         </div>
         <div class="clear"></div>
      </section>
      <hr />
      <footer> <p> Ejemplo de importar archivos en PHP </p> </footer>
   </div>
 </body>
</html>
```

twitter.php:

```
<!DOCTYPE>
<html>
 <head>
    <title> Ejemplo importar archivos </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link type="text/css" href="css/style.css" rel="stylesheet" />
 </head>
 <body>
    <div id="contenedor">
       <header>
          <h1> Redes sociales más populares </h1>
       </header>
       <nav>
          <ul>
             <li  class="google_mas">  <a  href="google_more.php">  Google+
</a> </li>
             <li  class="facebook"> <a href="facebook.php"> Facebook </a>
</li>
             <li  class="twitter">  <a  href="twitter.php">  Twitter  </a>
</li>
          </ul>
          <div class="clear"></div>
       </nav>
       <section>
          <div class="imagen_logo">
             <img src="images/twitter_logo.jpg" alt="Twitter" />
          </div>
          <div class="descripcion">
             <h2> Twitter </h2>
             <p> Twitter es un servicio de microblogging, con sede en San
Francisco, California, con filiales en San Antonio Texas y Boston
(Massachusetts) en Estados Unidos. Twitter, Inc. fue creado originalmente
en California, pero está bajo la jurisdicción de Delaware desde 2007.7
Desde que Jack Dorsey lo creó en marzo de 2006, y lo lanzó en julio del
mismo año, la red ha ganado popularidad mundialmente y se estima que
tiene más de 200 millones de usuarios, generando 65 millones de tuits al
día y maneja más de 800.000 peticiones de búsqueda diarias.1 Ha sido
apodado como el "SMS de Internet".8 Entre sus usuarios se destacan
grandes figuras públicas, como Barack Obama, actores como Danny DeVito,
músicos del estilo de Lady Gaga y Justin Bieber y otros mundialmente
conocidos. </p>
```

```
            <p> La red permite enviar mensajes de texto plano de corta
longitud, con un máximo de 140 caracteres, llamados tuits, que se
muestran en la página principal del usuario. Los usuarios pueden
suscribirse a los tuits de otros usuarios – a esto se le llama "seguir" y
a los usuarios abonados se les llama "seguidores",9 "followers" y a veces
tweeps10 ('Twitter' + 'peeps', seguidores novatos que aún no han hecho
muchos tweets). Por defecto, los mensajes son públicos, pudiendo
difundirse privadamente mostrándolos únicamente a unos seguidores
determinados. Los usuarios pueden tuitear desde la web del servicio, con
aplicaciones oficiales externas (como para teléfonos inteligentes), o
mediante el Servicio de mensajes cortos (SMS) disponible en ciertos
países.11 Si bien el servicio es gratis, acceder a él vía SMS comporta
soportar tarifas fijadas por el proveedor de telefonía móvil. </p>
            <p>       Extraído       de:       <a       target="_blank"
href="https://es.wikipedia.org/wiki/Twitter"> Wikipedia </a> </p>
          </div>
          <div class="clear"></div>
       </section>
       <hr />
       <footer> <p> Ejemplo de importar archivos en PHP </p> </footer>
    </div>
 </body>
</html>
```

Podemos probarlo iniciando el xampp y visitando la dirección:

```
https://localhost/ejemplo_importar_archivos/
```

Bueno, ya tenemos nuestro sitio armado, ahora bien, sucede que si notaste bien hay redundancia de código en las cuatro páginas. Por ejemplo la cabecera, la botonera y el pie de página, también el contenido de la etiqueta **head**. Entonces si de pronto el cliente nos dice que hay que modificar cosas como agregar un nuevo enlace en las redes sociales, tendremos que abrir los cuatro archivos y modificar por igual. ¿Y si tuviésemos veinte páginas? Una locura.

Sin embargo en PHP existen cuatro funciones que nos permitirán importar archivos dentro de un .php. **include**, **require**, **include_one** y **require_once**.

Entonces volviendo a nuestra página, lo único que no se repite entre las cuatro es el contenido de la etiqueta de la etiqueta **section**, por tanto vamos a partir el código html en tres partes. Por un lado todo lo que está por encima de **section**, luego el **section** mismo y finalmente lo que está por debajo. Así que vamos a crear dos archivos dentro de la carpeta **includes**, uno llamado **_header.php** y otro llamado **_footer.php**.

Dentro de **_header.php** copiaremos el siguiente código:

```
<!DOCTYPE>
<html>
 <head>
    <title> Ejemplo importar archivos </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link type="text/css" href="css/style.css" rel="stylesheet" />
 </head>
 <body>
    <div id="contenedor">
       <header>
          <h1> Redes sociales más populares </h1>
       </header>
       <nav>
          <ul>
             <li  class="google_mas"> <a  href="google_more.php"> Google+
</a> </li>
             <li class="facebook"> <a href="facebook.php"> Facebook </a>
</li>
             <li  class="twitter">  <a  href="twitter.php">  Twitter  </a>
</li>
          </ul>
          <div class="clear"></div>
       </nav>
```

Y dentro de **footer.php**:

```
          <hr />
         <footer> <p> Ejemplo de importar archivos en PHP </p> </footer>
      </div>
   </body>
</html>
```

Y ahora dentro de index.php reemplazaremos el código por esto:

```
<?php include 'includes/_header.php'; ?>
<section>
    <div class="descripcion">
       <h2> Redes sociales </h2>
       <p> Una red social es una forma de representar una estructura
social, asignándole un grafo, si dos elementos del conjunto de actores
(tales como individuos u organizaciones) están relacionados de acuerdo a
algún criterio (relación profesional, amistad, parentesco, etc.) entonces
se construye una línea que conecta los nodos que representan a dichos
elementos. El tipo de conexión representable en una red social es una
relación diádica o lazo interpersonal, que se pueden interpretar como
relaciones de amistad, parentesco, laborales, entre otros. </p>
```

```
       <p>         Extraído         de:         <a         target="_blank"
href="https://es.wikipedia.org/wiki/Redes_sociales"> Wikipedia </a> </p>
    </div>
    <div class="clear"></div>
</section>
<?php include 'includes/_footer.php'; ?>
```

Como se ve el código ha quedado mucho más resumido y lo que que se repite en todas las páginas lo hemos puesto en archivos apartes. Así que podemos continuar modificando los otros tres archivos:

google_more.php:

```
<?php include 'includes/_header.php'; ?>
<section>
    <div class="imagen_logo">
       <img src="images/google_mas_logo.jpg" alt="Google" />
    </div>
    <div class="descripcion">
       <h2> Google+ </h2>
       <p> Google+ (pronunciado y a veces escrito Google Plus, a veces
abreviado como G+, en algunos países de lengua hispana pronunciado Google
Más) es un servicio de red social operado por Google Inc. El servicio,
lanzado el 28 de junio de 2011, está basado en HTML5. Los usuarios tienen
que ser mayores de 18 años de edad,2 para crear sus propias cuentas.
Google+ ya es la segunda red social más popular del mundo con
aproximadamente 343 millones de usuarios activos. </p>
       <p> Google+ integra los servicios sociales, tales como Google
Perfiles y Google Buzz, e introduce los nuevos servicios: Círculos,
Quedadas, Intereses y Mensajes.3 Google+ también estará disponible como
una aplicación de escritorio y como una aplicación móvil, pero sólo en
los sistemas operativos Android e iOS. Fuentes tales como The New York
Times lo han declarado el mayor intento de Google para competir con la
red social Facebook,4 la cual tenía más de 750 millones de usuarios en
2011.5 El 20 de septiembre de 2011, Google permitió la creación de
cuentas a usuarios con más de 18 años, con mejoras en sus extensiones de
videoconferencias.6 </p>
       <p>         Extraído         de:         <a         target="_blank"
href="https://es.wikipedia.org/wiki/Google%2B"> Wikipedia </a> </p>
    </div>
    <div class="clear"></div>
</section>
<?php include 'includes/_footer.php'; ?>
```

facebook.php:

```
<?php include 'includes/_header.php'; ?>
<section>
    <div class="imagen_logo">
       <img src="images/facebook_logo.jpg" alt="Facebook" />
    </div>
    <div class="descripcion">
       <h2> Facebook </h2>
       <p> Facebook (NASDAQ: FB) es una empresa creada por Mark
Zuckerberg y fundada junto a Eduardo Saverin, Chris Hughes y Dustin
Moskovitz consistente en un sitio web de redes sociales. Originalmente
```

```
era un sitio para estudiantes de la Universidad de Harvard, pero actualmente está abierto a cualquier persona que tenga una cuenta de correo electrónico. Los usuarios pueden participar en una o más redes sociales, en relación con su situación académica, su lugar de trabajo o región geográfica. </p>
      <p> Ha recibido mucha atención en la blogosfera y en los medios de comunicación al convertirse en una plataforma sobre la que terceros pueden desarrollar aplicaciones y hacer negocio a partir de la red social. </p>
      <p> A mediados de 2007 lanzó las versiones en francés, alemán y español traducidas por usuarios de manera no remunerada,6 principalmente para impulsar su expansión fuera de Estados Unidos, ya que sus usuarios se concentran en Estados Unidos, Canadá y Reino Unido. Facebook cuenta con más de 900 millones de miembros, y traducciones a 70 idiomas.7 8 En octubre de 2012, Facebook llegó a los 1,000 millones de usuarios, de los cuáles hay más de 600 millones de usuarios móviles. Brasil, India, Indonesia, México y Estados Unidos son los países con el mayor número de usuarios.9 </p>
      <p> Su infraestructura principal está formada por una red de más de 50 000 servidores que usan distribuciones del sistema operativo GNU/Linux usando LAMP.10 </p>
      <p> El 9 de abril de 2012, se anunció que Facebook adquirió Instagram por mil millones de dólares.11 </p>
      <p>        Extraído        de:        <a        target="_blank"
href="https://es.wikipedia.org/wiki/Facebook"> Wikipedia </a> </p>
    </div>
    <div class="clear"></div>
</section>
<?php include 'includes/_footer.php'; ?>
```

twitter.php:

```
<?php include 'includes/_header.php'; ?>
<section>
    <div class="imagen_logo">
       <img src="images/twitter_logo.jpg" alt="Twitter" />
    </div>
    <div class="descripcion">
       <h2> Twitter </h2>
       <p> Twitter es un servicio de microblogging, con sede en San Francisco, California, con filiales en San Antonio Texas y Boston (Massachusetts) en Estados Unidos. Twitter, Inc. fue creado originalmente en California, pero está bajo la jurisdicción de Delaware desde 2007.7 Desde que Jack Dorsey lo creó en marzo de 2006, y lo lanzó en julio del mismo año, la red ha ganado popularidad mundialmente y se estima que tiene más de 200 millones de usuarios, generando 65 millones de tuits al día y maneja más de 800.000 peticiones de búsqueda diarias.1 Ha sido apodado como el "SMS de Internet".8 Entre sus usuarios se destacan grandes figuras públicas, como Barack Obama, actores como Danny DeVito, músicos del estilo de Lady Gaga y Justin Bieber y otros mundialmente conocidos. </p>
       <p> La red permite enviar mensajes de texto plano de corta longitud, con un máximo de 140 caracteres, llamados tuits, que se muestran en la página principal del usuario. Los usuarios pueden suscribirse a los tuits de otros usuarios – a esto se le llama "seguir" y a los usuarios abonados se les llama "seguidores",9 "followers" y a veces tweeps10 ('Twitter' + 'peeps', seguidores novatos que aún no han hecho
```

```
muchos tweets). Por defecto, los mensajes son públicos, pudiendo
difundirse privadamente mostrándolos únicamente a unos seguidores
determinados. Los usuarios pueden tuitear desde la web del servicio, con
aplicaciones oficiales externas (como para teléfonos inteligentes), o
mediante el Servicio de mensajes cortos (SMS) disponible en ciertos
países.11 Si bien el servicio es gratis, acceder a él vía SMS comporta
soportar tarifas fijadas por el proveedor de telefonía móvil. </p>
      <p>        Extraído        de:        <a        target="_blank"
href="https://es.wikipedia.org/wiki/Twitter"> Wikipedia </a> </p>
    </div>
    <div class="clear"></div>
</section>
<?php include 'includes/_footer.php'; ?>
```

De este modo nosotros estamos aprendiendo que una de las cosas más importantes en la creación de aplicaciones web, la reutilización de código, no repetir cinco mil veces lo mismo.

Si ahora el cliente nos pide que debemos agregar, modificar o quitar algún contenido lo haremos en un archivo y no en cuatro o cincuenta.

Descargar ejemplo

Diferencias entre include, require, include_once y require_once

Por empezar todas tienen el mismo fin, importar un archivo externo, ya sea un **.php**, **.html**,**.xml**, etc. Comencemos:

include

Esta función intentará importar un archivo, ahora bien, puede ser que ese archivo no exista porque lo han borrado o está mal escrito, en ese caso nos devolverá un error de tipo **Warning**. Esto significa que al no encontrar el archivo habrá una advertencia pero el script continuará. Por ejemplo:

```
<?php
 include 'archivo_que_no_existe.php';
 echo 'La vida continua';
?>
```

Ahora bien, los tipos de errores independientemente de que sean **Notice**, **Warning** o **Fatal error** no deberían estar. En este caso el **include** no romperá el script pero más adelante puede pagarse caro. Por ejemplo:

```
<?php
 include 'archivo_que_no_existe.php';
 funcionQueNuncaSeCargo();
```

```
 echo 'La vida continua';
?>
```

En este caso en la primer línea intenta cargar un archivo donde se supone que existe una función, el archivo no existe, y como se cargó con **include** no mostrará un **Warning**, el script sigue, pero en la siguiente línea se intenta llamar a una función que tampoco existe, ya que se supone que esa función estaba en el archivo. Y ahí sí, CHAU, se rompió el script definitivamente mostrando por pantalla un **Fatal error** y la última línea nunca se ejecuta.

require

Esta función es más estricta, si el archivo no lo encuentra tira un **Fatal error** y ahí se rompe el script:

```
<?php
 require 'archivo_que_no_existe.php';
 echo 'La vida continua';
?>
```

En este caso en la primer línea se rompe el script y la segunda nunca se ejecuta.

include_once

Hará lo mismo que **include**, si no encuentra el archivo devolverá un **Warning**, la única diferencia es que la importación del archivo la hará una vez, aunque el archivo se llame muchas veces. Por ejemplo si hacemos esto:

```
<?php
 include 'archivo_externo.php';
 include 'archivo_externo.php';
?>
```

Esto cargará dos veces y si se llamará a esa función con ese archivo cincuenta veces se cargará cincuenta veces también.

Ahora si usamos **include_once**:

```
<?php
 include_once 'archivo_externo.php';
 include_once 'archivo_externo.php';
?>
```

En este caso el archivo se cargará en la primer línea, pero la segunda no hará nada, ya que ese archivo con ese nombre ya ha sido cargado previamente en la primer línea.

require_once

En esta caso sucederá lo mismo que **include_once**, **require_once** importará un archivo, y se vuelve a llamar a esta función con el mismo archivo no hará nada. Pero al igual que**require**, de no encontrar el archivo se romperá el script con un **Fatal error**.

En los cuatros casos si el archivo no es encontrado nos devolverá un error, **Warning** en el caso de **include** y **include_once** y **Fatal error** en el caso de **require** y **require_once**.

Introducción a formularios (métodos GET y POST)

Bueno, al iniciar estas publicaciones dije que PHP recibe peticiones y devuelve respuestas, ahora bien, esas peticiones pueden venir por distintos métodos. Estos métodos pueden ser dos GET o POST, en la actualidad también existen otros más, pero sólo veremos los nombrados.

Ambos métodos tienen la capacidad de venir acompañados de datos, de variables, por ejemplo en los formularios los usuarios llenan los campos y esos valores son enviados al servidor empaquetados y con un nombre para hacer referencia.

Método GET

Hasta ahora todos los ejemplos que vimos, entre petición y respuesta fueron a través de este método, GET. Este método tiene la capacidad de recuperar valores por la url. Si tuviésemos una url como ésta:

```
https://localhost/proyecto/index.php
```

Podríamos pasarle variable a través de ? (signo de interrogación) variable1=valor&variable2=valor. Por ejemplo:

```
https://localhost/proyecto/index.php?fruta=manzana&color=rojo
```

Y luego recuperar esos valores desde PHP con:

```
<?php
 $fruta = $_GET['fruta'];
 $color = $_GET['color'];
 echo "La fruta es $fruta y es de color $color";
?>
```

Lo que nos devolverá por pantalla:

```
La fruta es manzana y es de color rojo
```

Como se ve en el ejemplo a través de la variable global **$_GET** podemos recuperar los valores que se pasan en la url.

Ahora, para probar cómo funciona esto a través de los formularios vamos a hacer lo siguiente. Vamos a crear un proyecto nuevo para buscar empleados por legajos.

Así que vamos a crear una carpeta dentro de htdocs llamada **buscar_legajos**. Y dentro de esa carpeta crearemos dos archivos .php, uno llamado **index.php** y otro **empleados.php**.

Por empezar, como aun no hemos trabajado con bases de datos, por tanto el archivo empleados.php será un array con una lista de empleados:

```
<?php
 $empleados = array(
    '10001' => array(
       'nombre' => 'Alejandro',
       'apellido' => 'Calvo',
       'sector' => 'Diseño'
    ),
    '10002' => array(
       'nombre' => 'Ignacio',
       'apellido' => 'Celestino',
       'sector' => 'Programación'
    ),
    '10003' => array(
       'nombre' => 'Marina',
       'apellido' => 'Oliver',
       'sector' => 'DB Master'
    )
);
?>
```

Y dentro del archivo index.php vamos a crear un formulario con un campo para ingresar el número de legajo:

```
<!DOCTYPE>
<html>
 <head>
    <title> Buscar empleados </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
    <form method="get" action="index.php">
       <label> Legajo: </label>
       <input type="text" name="legajo" required="required" />
       <input type="submit" value="Buscar" />
    </form>
 </body>
</html>
```

Bueno, por empezar tendremos un código html con un formulario y dentro un campo para ingresar el legajo y un botón que al pulsarlo buscará un empleado con ese legajo. El formulario tendrá dos atributos, el primero, **method**, es justamente la forma en que hará la petición. Mientras que el segundo, **action**, es adónde irá la petición, en nuestro caso es la misma página donde se encuentra el formulario, pero podría ser cualquier otra.

Ahora, antes de imprimir el html deberíamos crear un código php en la cabecera que verifique si se está haciendo una búsqueda y de así serlo mostrar el resultado de la búsqueda:

```
<?php
//Soporta caracteres especiales como tildes, eñes, etcétera.
header('Content-Type: text/html; charset=utf-8');
//Carga el array con la lista de los empleados.
require 'empleados.php';
//Esta variable verifica si se está realizando una búsqueda.
$legajo = (isset($_GET['legajo'])) ? $_GET['legajo'] : null;
//Esta variable guardará el resultado de la búsqueda.
$empleado = null;
//Si existe un legajo, es porque se está realizando una búsqueda.
if ($legajo) {
   //Limpiamos los espacios en blanco que pudo haber dejado el usuario.
   $legajo = trim($legajo);
   //Busca que exista un empleado con ese legajo.
   if (isset($empleados[$legajo])) {
      //Guarda la posición con los datos del empleado.
      $empleado = $empleados[$legajo];
   }
}
?>
```

Bien, ahora vamos a analizar el código porque puede resultar confuso al principio:

```
header('Content-Type: text/html; charset=utf-8');
```

Esta es la cabecera para definir el tipo de codificación, servirá por ejemplo para imprimir caracteres especiales con PHP.

```
require 'empleados.php';
```

Importamos el array con los datos de los empleados para realizar la búsqueda.

```
$legajo = (isset($_GET['legajo'])) ? $_GET['legajo'] : null;
```

Verificamos si existe una búsqueda a través de la variable ‘legajo’ a través de la función **isset()**, si existe esa variable, osea si el usuario envió el formulario. Para eso utilizamos un operador ternario. El operador ternario es un if:

```
(condicion) ? si se cumple : si no se cumple
```

Como se ve en el ejemplo dentro de la variable se guardará el resultado de la condición. Si existe la variable ‘legajo’ guardará el valor de la misma, sino la dejará en **null**.

```
$empleado = null;
```

Creará una variable que por defecto será **null**, luego de realizar la búsqueda si se encuentra al empleado lo guardará en esta variable.

```
if ($legajo) {
```

```
   $legajo = trim($legajo);
   if (isset($empleados[$legajo])) {
      $empleado = $empleados[$legajo];
   }
}
```

Preguntamos si el legajo existe, osea si se ha enviado el formulario y finalmente verificamos si en el array de empleados existe una posición con ese legajo y de cumplirse la condición guardará esa posición en la variable **$empleado**.

Bien, sólo nos falta imprimir el resultado de la búsqueda en el código html:

```
<div>
   <?php
      if ($legajo) {
         if ($empleado) {
            echo "Nombre: {$empleado['nombre']} {$empleado['apellido']} -
Sector: {$empleado['sector']}";
         } else {
            echo 'No existe un empleado con ese legajo.';
         }
      }
   ?>
 </div>
```

Aquí preguntamos si existe un legajo ingresado por el usuario con la variable **$legajo**, y de ser así mostramos el resultado de la búsqueda con la variable **$empleado**, osea si encontró el empleado dentro de ésta estarán los datos del mismo y serán mostrados por pantalla.

Entonces el código de index.php quedaría así:

```
<?php
//Soporta caracteres especiales como tildes, eñes, etcétera.
header('Content-Type: text/html; charset=utf-8');
//Carga el array con la lista de los empleados.
require 'empleados.php';
//Esta variable verifica si se está realizando una búsqueda.
$legajo = (isset($_GET['legajo'])) ? $_GET['legajo'] : null;
//Esta variable guardará el resultado de la búsqueda.
$empleado = null;
//Si existe un legajo, es porque se está realizando una búsqueda.
if ($legajo) {
   //Limpiamos los espacios en blanco que pudo haber dejado el usuario.
   $legajo = trim($legajo);
   //Busca que exista un empleado con ese legajo.
   if (isset($empleados[$legajo])) {
      //Guarda la posición con los datos del empleado.
      $empleado = $empleados[$legajo];
   }
}
?>
<!DOCTYPE>
<html>
 <head>
    <title> Buscar empleados </title>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
    <form method="get" action="index.php">
       <label> Legajo: </label>
       <input type="text" name="legajo" required="required" value="<?php
echo $legajo ?>" />
       <input type="submit" value="Buscar" />
    </form>
    <div>
       <?php
          if ($legajo) {
             if ($empleado) {
                echo             "Nombre:             {$empleado['nombre']}
{$empleado['apellido']} - Sector: {$empleado['sector']}";
             } else {
                echo 'No existe un empleado con ese legajo.';
             }
          }
       ?>
    </div>
 </body>
</html>
```

Ya podemos probar nuestro código yendo a:

```
https://localhost/buscar_legajos/
```

Y probar ingresando los valores 10001, 10002 y 10003 que son los que devuelve resultados.

Descargar ejemplo

Método POST

Ahora bien, si te has fijado cuando enviamos el formulario la página se recarga con la url y el **name** y valor del campo que enviamos, algo con este aspecto:

```
https://localhost/buscar_legajos/?legajo=10001
```

Esto es muy útil para un buscador, ya que luego el usuario puede copiar la url y pasársela a otro usuario y así mostrarle el resultado. Pero si por ejemplo debemos usar otro tipo de formulario, por ejemplo un login, entones no es buena idea que los datos que envía el usuario queden el nombre de usuario y la contraseña en la url, esto implica un error de seguridad, por tanto nosotros necesitamos ocultar esos datos. Para ello debemos recurrir al método POST.

En este caso vamos a crear un nuevo proyecto dentro de htdocs llamado **login**, que obviamente no pegará en una base de datos (que veremos más adelante) sino en una simulación de datos de nombre de usuario y contraseña.

Y ahora dentro de **login** vamos crear dos archivos, uno llamado **index.php** y otro llamado **datos_usuarios.php**.

Dento de datos_usuarios.php simplemente vamos a crear dos contantes una llamada **USUARIO** y contra **CONTRASENA**:

```
<?php
   define('USUARIO', 'pepito');
   define('CONTRASENA', '1234');
?>
```

Y dentro de index.php vamos a copiar este código html:

```
<!DOCTYPE>
<html>
 <head>
    <title> Inicio se sesión </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
    <form method="post" action="index.php">
       <label> Nombre de usuario: </label>
       <br />
       <input type="text" name="usuario" required="required" />
       <br />
       <label> Contraseña: </label>
       <br />
       <input type="password" name="contrasena" required="required" />
       <br />
       <input type="submit" value="Ingresar" />
 </form>
 </body>
</html>
```

Por empezar este formulario lo enviaremos mediante el método POST, en este caso, cuando la página se recargue con las variables **usuario** y **contrasena** y sus valores, estos datos estarán ocultos en la cabecera de la página y no se verán en la url, lo cual lo hace más seguro al proteger estos datos.

Ahora sólo nos falta agregar la cabecera PHP, para que cuando se envíe el formulario se verifique si el nombre de usuario y la contraseña son correctos:

```
<?php
//Importamos los datos de usuario con el nombre y la contraseña.
require_once 'datos_usuarios.php';
//Verifica si la petición viene por POST, osea si el usuario envió el
formulario.
$por_post = ($_SERVER['REQUEST_METHOD'] == 'POST');
if ($por_post) {
```

```
   //Recupera los valores de los datos enviados por el usuario.
   $usuario = $_POST['usuario'];
   $contrasena = $_POST['contrasena'];
   //Verica si los datos ingresados corresponden al nombre de usuario y
la contraseña.
   if ($usuario == USUARIO and $contrasena == CONTRASENA) {
      $login_correcto = true;
   } else {
      $login_correcto = false;
   }
}
?>
```

Ok, vamos a analizar el código:

```
require_once 'datos_usuarios.php';
```

Importamos el archivo donde está el nombre de usuario y la contraseña, para simular que lo está buscando de una base de datos, nada complejo.

```
$por_post = ($_SERVER['REQUEST_METHOD'] == 'POST');
```

Esta línea nos permitirá saber si la petición del usuario ha venido por POST, lo cual significa que el usuario ha enviado el formulario. La variable globlal **$_SERVER** es un array que nos provee de información importante, si querés saber más a fondo de ésta, puede chequear el manual. En este caso, a través del índice 'REQUEST_METHOD' nosotros podemos saber si la petición ha venido por GET o por POST. Por tanto dentro de la variable **$por_post** se guardará **true** si se está intentando loguear o no el usuario.

```
$usuario = $_POST['usuario'];
$contrasena = $_POST['contrasena'];
```

Aquí simplemente guardamos en variables los valores que llegaron del usuario.

```
if ($usuario == USUARIO and $contrasena == CONTRASENA) {
   $login_correcto = true;
} else {
   $login_correcto = false;
}
```

Y aquí verificamos si los datos coinciden con el usuario y la contraseña, si es correcto guardará dentro de la variable **$login_correcto true**, sino será **false**.

Finalmente el código quedará así:

```
<?php
//Importamos los datos de usuario con el nombre y la contraseña.
require_once 'datos_usuarios.php';
//Verifica si la petición viene por POST, osea si el usuario envió el
formulario.
$por_post = ($_SERVER['REQUEST_METHOD'] == 'POST');
if ($por_post) {
   //Recupera los valores de los datos enviados por el usuario.
```

```
   $usuario = $_POST['usuario'];
   $contrasena = $_POST['contrasena'];
   //Verica si los datos ingresados corresponden al nombre de usuario y
la contraseña.
   if ($usuario == USUARIO and $contrasena == CONTRASENA) {
      $login_correcto = true;
   } else {
      $login_correcto = false;
   }
}
?>
<!DOCTYPE>
<html>
 <head>
    <title> Inicio se sesión </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
    <?php if ($por_post and $login_correcto): //Verifica si el usuario
envió el formulario y el login es correcto. ?>
       <div> Bienvenido/a. </div>
    <?php else: ?>
       <form method="post" action="index.php">
          <label> Nombre de usuario: </label>
          <br />
          <input type="text" name="usuario" required="required" />
          <br />
          <label> Contraseña: </label>
          <br />
          <input type="password" name="contrasena" required="required" />
          <br />
          <input type="submit" value="Ingresar" />
          <?php if ($por_post and !$login_correcto): //Verifica si el
usuario envió el formulario y el login es incorrecto. ?>
             <div> El usuario o la contraseña son incorrectos. </div>
          <?php endif; ?>
       </form>
    <?php endif; ?>
 </body>
</html>
```

Por empezar usaremos los condicionales de tipo tag, que son muy útiles para mezclar PHP con HTML. Habrá un condicional que verificará si el usuario ha enviado el usuario y si el nombre de usuario y la contraseña son correctos:

```
<?php if ($por_post and $login_correcto): ?>
```

De ser así mostrará un mensaje por pantalla:

```
<div> Bienvenido/a. </div>
```

De lo contrario mostrará el formulario y dentro del mismo que verificará si se ha enviado el formulario pero el nombre de usuario y la contraseña son incorrectos:

```
<?php if ($por_post and !$login_correcto): ?>
```

Descargar ejemplo

Elementos de formulario

En la anterior publicación conocimos el envío de formularios y los dos métodos que nos permiten enviarlos al servidor, GET y POST. En esa ocasión habíamos usado dos elementos, **input text**, para ingresar texto e **input password** para las contraseñas. En esta ocasión profundizaremos un poco más sobre los elementos de formulario, aunque si bien esto pertenece más a un tema de HTML más que a PHP es importante entender bien su funcionamiento y como llegan los valores ingresados por el usuario al servidor.

Para ello vamos a crear un pequeño formulario con algunos de los campos más usados:

```
<!DOCTYPE>
<html>
 <head>
    <title> Formulario </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
    <form id="formulario" method="post" action="index.php">
       <table>
          <tr>
             <td> <label> Nombre y apellido </label> </td>
             <td>      <input      type="text"      name="nombre_apellido"
required="required" /> </td>
          </tr>
          <tr>
             <td> <label> Sexo </label> </td>
             <td>
                <input    type="radio"    name="sexo"    value="Femenino"
required="required" /> Femenino
                <input    type="radio"    name="sexo"    value="Masculino"
required="required" /> Masculino
             </td>
          </tr>
          <tr>
             <td> <label> Nacionalidad </label> </td>
             <td>
                <select name="nacionalidad" required="required">
                   <option value=""> --- </option>
                   <option value="Argentina"> Argentina </option>
                   <option value="Bolivia"> Bolivia </option>
                   <option value="Brasil"> Brasil </option>
                   <option value="Chile"> Chile </option>
                   <option value="Paraguay"> Paraguay </option>
                   <option value="Uruguay"> Uruguay </option>
                </select>
             </td>
          </tr>
```

```
          <tr>
             <td> <label> Intereses </label> </td>
             <td>
                <input type="checkbox" name="intereses[]" value="Cine" />
Cine
                <input         type="checkbox"         name="intereses[]"
value="Deportes" /> Deportes
                <input         type="checkbox"         name="intereses[]"
value="Deportes" /> Internet
                <input  type="checkbox"  name="intereses[]"  value="Libros"
/> Libros
                <input  type="checkbox"  name="intereses[]"  value="Música"
/> Música
             </td>
          </tr>
          <tr>
             <td> <label> Acerca de vos </label> </td>
             <td>
                <textarea    name="acerca_de_vos"    rows="5"    cols="50"
required="required"></textarea>
             </td>
          </tr>
       </table>
       <input type="submit" value="Enviar" />
    </form>
 </body>
</html>
```

Esto le dará un aspecto sencillo pero que nos servirá de ejemplo:

Ahora vamos a analizar los elementos. Comencemos.

Input text

```
<input type="text" name="nombre_apellido" required="required" />
```

Este elemento ya lo vimos, y es muy fácil de entender, una vez enviado el formulario llegará al servidor el cual podremos recuperar por su **name**, como todo elemento de formulario, en este caso "nombre_apellido".

Input radio

```
<input type="radio" name="sexo" value="Femenino" required="required" />
<input type="radio" name="sexo" value="Masculino" required="required" />
```

Los elementos de tipo **input radio** permiten que el usuario envíe un único valor dentro de un grupo. Si te has fijado bien, para definir grupos debemos usar el mismo **name** para todos los que formen justamente ese grupo. Además debemos ingresar de forma obligatoria un segundo atributo llamado **value** que será el valor con el que llegue ese valor al servidor. En este ejemplo de elegir el primero el valor será "Femenino", sino el segundo, "Masculino".

Select

```
<select name="nacionalidad" required="required">
   <option value=""> --- </option>
   <option value="Argentina"> Argentina </option>
   <option value="Bolivia"> Bolivia </option>
   <option value="Brasil"> Brasil </option>
   <option value="Chile"> Chile </option>
   <option value="Paraguay"> Paraguay </option>
   <option value="Uruguay"> Uruguay </option>
</select>
```

Los elementos de tipo **select**, también conocidos en otros lenguajes como combobox, al igual que los **input radio** nos permiten seleccionar una opción dentro de un grupo, aunque estos son más útiles cuando se tienen muchas opciones. Es importante aclarar que el valor que viajará al servidor, osea el que elija el usuario, es obviamente el del atributo **value** y no el que se le muestra al usuario. Osea que si por ejemplo uno de los **option** fuese:

```
<option value="py"> Paraguay </option>
```

El valor que llegará al servidor será "py" y no "Paraguay".

Input checbox

```
<input type="checkbox" name="intereses[]" value="Cine" /> Cine
<input type="checkbox" name="intereses[]" value="Deportes" /> Deportes
<input type="checkbox" name="intereses[]" value="Deportes" /> Internet
<input type="checkbox" name="intereses[]" value="Libros" /> Libros
<input type="checkbox" name="intereses[]" value="Música" /> Música
```

Los **input checkbox** a diferencia de los **input radio** y **select** permiten elegir varias opciones dentro de un grupo. Además, como en este caso, si necesitamos enviar mútiples opciones, el valor del **name** debemos usarlo con corchetes de apertura y cierre al final para

que al llegar al servidor y PHP lo recupere como un array, en donde cada posición sea cada valor de los que han sido seleccionados por el usuario.

Textarea

```
<textarea name="acerca_de_vos" rows="5" cols="50" required="required"></textarea>
```

Los campos de tipo **textarea**, al igual que los **input text**, son campos de texto libre, aunque claro, estos están pensados para ingresos de mucho texto.

También existen otros campos que no están en el ejemplo, pero los nombraré a continuación:

Input file

```
<input type="file" name="imagen" />
```

Estos campos nos permiten adjuntar archivos, pero lo veremos más adelante ya que merece un capítulo aparte.

Input password

```
<input type="password" name="contrasena" />
```

Estos permiten ingresar datos privados como contraseñas, y cada caracteres será invisible para protegerlo visualmente.

Input hidden

```
<input type="hidden" name="campo_oculto" value="Este es un valor oculto" />
```

Se utilizan para valores que estarán ocultos en el formulario, y su fin es que el usuario no tena que llenarlos. Más adelante veremos que son muy útiles.

Input button

```
<input type="button" value="Enviar" />
```

Es un botón, como **input submit**, pero a diferencia de éste, no sirve para enviar el formulario, sino que sirve para otro tipo de cosas, por ejemplo se le puede aplicar funcionalidad Javascript.

Input reset

```
<input type="reset" value="Limpiar campos" />
```

También es un botón, que al pulsarlo vaciará todos los datos que haya ingresado el usuario en el formulario donde está contenido. Este elemento no suele utilizarse mucho.

Input image

```
<input type="image" value="Enviar" src="imagen.jpg" />
```

También permite enviar el formulario, pero es un botón que da la posibilidad de cargarle una imagen a través del atributo **src**.

Input submit

```
<input type="submit" value="Enviar" />
```

Al pulsar este botón el formulario será enviado al servidor.

Además con la salida de Html5 se han integrado nuevos campos, si querés verlos podés visitar esta publicación: Elementos de formulario en html5.

Bueno siguiendo con el ejemplo que estábamos haciendo, vamos a crear la cabecera de la página, en donde recibiremos los campos que fueron enviados del formulario:

```
<?php
header('Content-Type: text/html; charset=utf-8');
$por_post = ($_SERVER['REQUEST_METHOD'] == 'POST');
if ($por_post) {
   $nombre_apellido = $_POST['nombre_apellido'];
   $sexo = $_POST['sexo'];
   $nacionalidad = $_POST['nacionalidad'];
   $intereses  =  (isset($_POST['intereses']))  ?  $_POST['intereses']  :
null;
   $acerca_de_vos = $_POST['acerca_de_vos'];
}
?>
```

Bueno, como en el ejemplo anterior definimos una cabecera utf-8 para que el PHP soporte caracteres especiales. Además preguntamos si hay una petición por POST, para comprobar si el usuario está enviando el formulario y de ser así guardamos las variables de los campos.

Notar que al recuperar los intereses debemos utilizar el operador ternario preguntando si existe una variable $_POST[‘intereses’]:

```
$intereses = (isset($_POST['intereses'])) ? $_POST['intereses'] : null;
```

Esto se debe a que si el usuario no ha seleccionado ninguno de los checkbox no llegará ninguna variable POST con ese nombre. Además recordar que si en cambio el usuario selecciona uno o más de los checkbox esta variable será un arrray.

Ahora vamos a terminar nuestro ejemplo, haciendo lo siguiente: si el usuario llena el formulario y pulsa el botón de enviar deben mostrarse los datos que ingresó:

```
<?php
header('Content-Type: text/html; charset=utf-8');
$por_post = ($_SERVER['REQUEST_METHOD'] == 'POST');
if ($por_post) {
   $nombre_apellido = $_POST['nombre_apellido'];
   $sexo = $_POST['sexo'];
   $nacionalidad = $_POST['nacionalidad'];
   $intereses  =  (isset($_POST['intereses']))  ?  $_POST['intereses']  :
null;
   $acerca_de_vos = $_POST['acerca_de_vos'];
}
?>
<!DOCTYPE>
<html>
 <head>
    <title> Formulario </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
    <?php if ($por_post): ?>
       <ul>
          <li>  <strong>  Nombre  y  apellido:  </strong>  <?php  echo
$nombre_apellido ?> </li>
          <li> <strong> Sexo: </strong> <?php echo $sexo ?> </li>
          <li> <strong> Nacionalidad: </strong> <?php echo $nacionalidad
?> </li>
          <?php  if($intereses):  ?>  <li>  <strong>  Intereses:  </strong>
<?php echo implode(' - ', $intereses) ?> </li> <?php endif; ?>
       </ul>
       <div>
          <?php echo nl2br($acerca_de_vos) ?>
       </div>
    <?php else: ?>
       <form id="formulario" method="post" action="index.php">
          <table>
             <tr>
                <td> <label> Nombre y apellido: </label> </td>
                <td>     <input     type="text"     name="nombre_apellido"
required="required" /> </td>
             </tr>
             <tr>
                <td> <label> Sexo: </label> </td>
                <td>
                   <input   type="radio"   name="sexo"   value="Femenino"
required="required" /> Femenino
                   <input   type="radio"   name="sexo"   value="Masculino"
required="required" /> Masculino
                </td>
             </tr>
```

```
            <tr>
               <td> <label> Nacionalidad: </label> </td>
               <td>
                  <select name="nacionalidad" required="required">
                     <option value=""> --- </option>
                     <option value="Argentina"> Argentina </option>
                     <option value="Bolivia"> Bolivia </option>
                     <option value="Brasil"> Brasil </option>
                     <option value="Chile"> Chile </option>
                     <option value="Paraguay"> Paraguay </option>
                     <option value="Uruguay"> Uruguay </option>
                  </select>
               </td>
            </tr>
            <tr>
               <td> <label> Intereses: </label> </td>
               <td>
                  <input type="checkbox" name="intereses[]" value="Cine"
/> Cine
                  <input       type="checkbox"       name="intereses[]"
value="Deportes" /> Deportes
                  <input       type="checkbox"       name="intereses[]"
value="Deportes" /> Internet
                  <input       type="checkbox"       name="intereses[]"
value="Libros" /> Libros
                  <input       type="checkbox"       name="intereses[]"
value="Música" /> Música
               </td>
            </tr>
            <tr>
               <td> <label> Acerca de vos: </label> </td>
               <td>
                  <textarea   name="acerca_de_vos"   rows="5"   cols="50"
required="required"></textarea>
               </td>
            </tr>
         </table>
         <input type="submit" value="Enviar" />
      </form>
   <?php endif; ?>
 </body>
</html>
```

Bueno vamos a analizar las nuevas líneas, en primer lugar verificamos si se ha enviado el formulario con:

```
<?php if ($por_post): ?>
```

Si así fue mostrará los datos que envió el usuario, de lo contrario será el formulario.

Los valores serán impresos por pantalla en una lista y el campo **textarea** con un **div**:

```
<ul>
   <li> <strong> Nombre y apellido: </strong> <?php echo $nombre_apellido
?> </li>
   <li> <strong> Sexo: </strong> <?php echo $sexo ?> </li>
```

```
   <li> <strong> Nacionalidad: </strong> <?php echo $nacionalidad ?>
</li>
   <?php if($intereses): ?> <li> <strong> Intereses: </strong> <?php echo
implode(' - ', $intereses) ?> </li> <?php endif; ?>
</ul>
<div>
   <?php echo nl2br($acerca_de_vos) ?>
</div>
```

Notar dos cosas. En primer lugar la forma en que mostramos los intereses, los checkbox que el usuario tildó. Usamos un **if** para verificar si existen los intereses, osea que si no ha tildado ninguno, la variable será **null** y salteará ese item. Segundo, la forma que mostramos los intereses, de haberlos, a través de la función **implode()**, que como vimos en una publicación pasada, convierte un array en un string separando sus posiciones con un caracteres específico. También podríamos haber recorrido el array con un **foreach** y mostrar cada posición, cada interés, uno abajo del otro.

Tal vez te preguntes también porque la variable **$acerca_de_vos** la mostramos con la función **nl2br()**, esta función convierte los saltos de línea en etiquetas **
**, debido a que HTML no los tiene en cuenta, o mejor dicho no los muestra como tal.

Esto nos mostrará por pantalla algo como esto:

- **Nombre y apellido:** Fernando Gaitán
- **Sexo:** Masculino
- **Nacionalidad:** Argentina
- **Intereses:** Deportes - Deportes - Libros - Música

Soy programador y me encanta PHP.

También me gustan el fútbol, los videojuegos y viajar.

Chau!

Descargar ejemplo

Validar formularios

Cuando hablamos de validaciones de formulario es inevitable pensar en Javascript, y esto tiene mucha lógica, ya que uno de los motivos por los cuales se ha creado este lenguaje es para evitar el envío de información inútil al servidor, sin embargo, siempre hay que tener presente que las validaciones en el navegador, aunque son necesarias para extender la funcionalidad del frontend, éste puede ser vulnerado fácilmente, ya que el usuario puede

tener Javascript desactivado o bien, puede tener conocimientos medios de web y saltearse estas validaciones.

Pero las validaciones del backend son una herramienta mucho más segura, por eso es importante que las validaciones con PHP, o cualquier otro lenguaje de servidor nunca falten, ya que es el punto fuerte para evitar la entrada de datos incorrectos por ejemplo en una base de datos.

En esta ocasión veremos un sencillo ejemplo. Así que vamos a crear una nueva carpeta dentro de htdocs llamada **validaciones**. Crearemos un archivo llamado **index.php** y otro llamado **validado.php** y a la misma altura una carpeta llamada **funciones** que dentro tendrá otro archivo llamado **validaciones.php**.

Dentro del archivo **index.php** vamos a copiar el siguiente código:

```
<!DOCTYPE>
<html>
 <head>
 <title> Formulario </title>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
    <form method="post" action="index.php">
       <label> Nombre </label>
       <br />
       <input type="text" name="nombre" />
       <br />
       <label> Edad </label>
       <br />
       <input type="text" name="edad" size="3" />
       <br />
       <label> E-mail </label>
       <br />
       <input type="text" name="email" />
       <br />
       <input type="submit" value="Enviar" />
    </form>
 </body>
</html>
```

Y dentro de **validado.php** lo siguiente:

```
<!DOCTYPE>
<html>
 <head>
 <title> Formulario </title>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
    <strong> Sus datos han sido enviados correctamente </strong>
 </body>
</html>
```

Este último no tendrá mucha lógica, será simplemente un mensaje de que el formulario ha sido validado correctamente.

Con respecto al anterior, al archivo **index.php**, como se ve en el ejemplo será un simple formulario con tres campos para el nombre, la edad y la dirección de e-mail, algo sencillo que a estas alturas te debería resultar familiar.

Ahora, el asunto es así, cuando se envíe el formulario deben validarse tres cosas: por empezar que el campo nombre no debe estar vacío, el campo de la edad debe ser un número entero y finalmente la dirección de e-mail debe tener un formato válido.

Así que vamos dentro de la carpeta funciones al archivo **validaciones.php** y crear las tres funciones correspondientes.

Empecemos por la validaciones del campo requerido, podríamos tener una función con este aspecto:

```
function validaRequerido($valor){
   if($valor == ''){
      return false;
   }else{
      return true;
   }
}
```

La misma recibirá un valor que será justamente lo que ingresó el usuario, preguntará si éste son comillas vacías, osea si no ingresó nada, y de ser así devolverá **false**, de lo contrario devolverá **true**, osea que está validado.

Ahora bien, esto tiene un pequeño problema, y es que si el usuario ingresa un campo sin llenar llegará al servidor una variable como ésta:

```
$nombre = '';
```

Pero también el usuario podría meter espacios y no escribir nada más:

```
$nombre = ' ';
```

(Notar el espacio dentro de las comillas)

Acá la validación fallará porque nuestra función pregunta si es comillas-comillas, y no comillas-espacio-comillas. Entonces para solucionar este problema usaremos la función **trim()** que eliminará las comillas de adelante y atrás. Así que vamos a reemplazar la función por:

```
function validaRequerido($valor){
   if(trim($valor) == ''){
      return false;
   }else{
      return true;
   }
```

```
}
```

Bueno, seguimos. Vamos a crear la validación para la edad, éste debe ser un número entero, así que vamos a usar una función que no habíamos visto hasta ahora que es la función **filter_var()**. Ésta recibirá tres parámetros, el primero será el valor a validar o filtrar, el segundo el tipo de validación y el tercero será un array con reglas opcionales. Éste último vamos a omitirlo momentáneamente. Esta función nos devolverá el valor que le pasamos, pero si éste no es válido entonces será **FALSE**:

```
function validarEntero($valor){
   if(filter_var($valor, FILTER_VALIDATE_INT) === FALSE){
      return false;
   }else{
      return true;
   }
}
```

Si te has fijado bien, el segundo parámetro que le hemos pasado a la función es una constante propia de PHP llamada **FILTER_VALIDATE_INT**, osea para los números enteros. Aunque esta validación no está completa, ya que la función va a validar que el valor que ingresen sea un entero, pero no tiene un rango, osea si el usuario pone que tiene 500 años, esto obviamente es incorrecto. Bueno, como dijimos antes la función **filter_var()** tiene un tercer parámetro opcional que es un array con filtros, así que vamos a modificar nuestra función por esto:

```
function validarEntero($valor, $opciones=null){
   if(filter_var($valor, FILTER_VALIDATE_INT, $opciones) === FALSE){
      return false;
   }else{
      return true;
   }
}
```

Como se ve, la función **validarEntero()** pasamos un segundo parámetro opcional que se pasará a **filter_var()**, este parámetro si lo salteamos será **null** y no se tendrá en cuenta.

Y finalmente una función para validar el email en donde también usaremos la función de PHP **filter_var()** sólo que en esta ocasión en lugar de usar **FILTER_VALIDATE_INT** usaremos **FILTER_VALIDATE_EMAIL**:

```
function validaEmail($valor){
   if(filter_var($valor, FILTER_VALIDATE_EMAIL) === FALSE){
      return false;
   }else{
      return true;
   }
}
```

Finalmente nuestro archivo **validaciones.php** quedará así:

```
<?php
 function validaRequerido($valor){
```

```
    if(trim($valor) == ''){
       return false;
    }else{
       return true;
    }
 }
 function validarEntero($valor, $opciones=null){
    if(filter_var($valor, FILTER_VALIDATE_INT, $opciones) === FALSE){
       return false;
    }else{
       return true;
    }
 }
 function validaEmail($valor){
    if(filter_var($valor, FILTER_VALIDATE_EMAIL) === FALSE){
       return false;
    }else{
       return true;
    }
 }
?>
```

Y ahora vamos a volver al archivo **index.php** para editar la cabecera:

```
<?php
//Definimos la codificación de la cabecera.
header('Content-Type: text/html; charset=utf-8');
//Importamos el archivo con las validaciones.
require_once 'funciones/validaciones.php';
//Guarda los valores de los campos en variables, siempre y cuando se haya
enviado el formulario, sino se guardará null.
$nombre = isset($_POST['nombre']) ? $_POST['nombre'] : null;
$edad = isset($_POST['edad']) ? $_POST['edad'] : null;
$email = isset($_POST['email']) ? $_POST['email'] : null;
//Este array guardará los errores de validación que surjan.
$errores = array();
//Pregunta si está llegando una petición por POST, lo que significa que
el usuario envió el formulario.
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
   //Valida que el campo nombre no esté vacío.
   if (!validaRequerido($nombre)) {
      $errores[] = 'El campo nombre es incorrecto.';
   }
   //Valida la edad con un rango de 3 a 130 años.
   $opciones_edad = array(
      'options' => array(
         //Definimos el rango de edad entre 3 a 130.
         'min_range' => 3,
         'max_range' => 130
      )
   );
   if (!validarEntero($edad, $opciones_edad)) {
      $errores[] = 'El campo edad es incorrecto.';
   }
   //Valida que el campo email sea correcto.
   if (!validaEmail($email)) {
      $errores[] = 'El campo email es incorrecto.';
```

```
   }
   //Verifica si ha encontrado errores y de no haber redirige a la página
con el mensaje de que pasó la validación.
   if(!$errores){
      header('Location: validado.php');
      exit;
   }
}
?>
```

Ahora vamos a explicar las líneas:

```
header('Content-Type: text/html; charset=utf-8');
```

Definimos el formato, nada nuevo.

```
require_once 'funciones/validaciones.php';
```

Importamos el archivo con las validaciones.

```
$nombre = isset($_POST['nombre']) ? $_POST['nombre'] : null;
$edad = isset($_POST['edad']) ? $_POST['edad'] : null;
$email = isset($_POST['email']) ? $_POST['email'] : null;
```

Estas variables guardarán los valores que ha enviado el usuario si es que ha enviado el formulario, sino se guardará **null**.

```
$errores = array();
```

Esta array guardará los errores de validación si es que hay. Luego nos servirá para verificar si el formulario está correcto.

```
if ($_SERVER['REQUEST_METHOD'] == 'POST')
```

Verificamos si el usuario ha enviado el formulario, osea que la petición está llegando por POST. Dentro de este **if** vamos a validar que los campos estén correctos.

```
//Valida que el campo nombre no esté vacío.
if (!validaRequerido($nombre)) {
   $errores[] = 'El campo nombre es incorrecto.';
}
//Valida la edad con un rango de 3 a 130 años.
$opciones_edad = array(
   'options' => array(
      //Definimos el rango de edad entre 3 a 130.
      'min_range' => 3,
      'max_range' => 130
   )
);
if (!validarEntero($edad, $opciones_edad)) {
   $errores[] = 'El campo edad es incorrecto.';
}
//Valida que el campo email sea correcto.
if (!validaEmail($email)) {
```

```
   $errores[] = 'El campo email es incorrecto.';
}
```

Aquí validamos los valores que ha enviado el formulario preguntando en cada validación si da **false**, de ser así guardamos un nuevo error de validación dentro de el array **$errores**.

Notar esta línea:

```
$opciones_edad = array(
   'options' => array(
      //Definimos el rango de edad entre 3 a 130.
      'min_range' => 3,
      'max_range' => 130
   )
);
```

Definimos el mínimo y máximo de rango, osea que la edad esté entre 3 y 130 años.

```
if(!$errores){
   header('Location: validado.php');
   exit;
}
```

Preguntamos si el array **$errores** está vacío, de ser así significa que no tienen errores (claro está), osea que el formulario está validado y redireccionamos con la función **header()** a la página **validado.php**. Con respecto a la función **exit** está nos permite finalizar el script, osea que todo lo que siga por delante será omitido.

Ahora, si el array **$errores** no está vacío, osea que hay errores, debemos mostrar los mismos dentro del código html:

```
<?php if ($errores): ?>
   <ul style="color: #f00;">
      <?php foreach ($errores as $error): ?>
         <li> <?php echo $error ?> </li>
      <?php endforeach; ?>
   </ul>
<?php endif; ?>
```

Aquí preguntamos si hay errores, y de haberlos los mostramos recorriendo el array **$errores** y los mostramos con una lista.

El código finalmente quedará así:

```
<?php
//Definimos la codificación de la cabecera.
header('Content-Type: text/html; charset=utf-8');
//Importamos el archivo con las validaciones.
require_once 'funciones/validaciones.php';
//Guarda los valores de los campos en variables, siempre y cuando se haya
enviado el formulario, sino se guardará null.
$nombre = isset($_POST['nombre']) ? $_POST['nombre'] : null;
$edad = isset($_POST['edad']) ? $_POST['edad'] : null;
```

```
$email = isset($_POST['email']) ? $_POST['email'] : null;
//Este array guardará los errores de validación que surjan.
$errores = array();
//Pregunta si está llegando una petición por POST, lo que significa que
el usuario envió el formulario.
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
   //Valida que el campo nombre no esté vacío.
   if (!validaRequerido($nombre)) {
      $errores[] = 'El campo nombre es incorrecto.';
   }
   //Valida la edad con un rango de 3 a 130 años.
   $opciones_edad = array(
      'options' => array(
         //Definimos el rango de edad entre 3 a 130.
         'min_range' => 3,
         'max_range' => 130
      )
   );
   if (!validarEntero($edad, $opciones_edad)) {
      $errores[] = 'El campo edad es incorrecto.';
   }
   //Valida que el campo email sea correcto.
   if (!validaEmail($email)) {
      $errores[] = 'El campo email es incorrecto.';
   }
   //Verifica si ha encontrado errores y de no haber redirige a la página
con el mensaje de que pasó la validación.
   if(!$errores){
      header('Location: validado.php');
      exit;
   }
}
?>
<!DOCTYPE>
<html>
 <head>
    <title> Formulario </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
    <?php if ($errores): ?>
       <ul style="color: #f00;">
          <?php foreach ($errores as $error): ?>
             <li> <?php echo $error ?> </li>
          <?php endforeach; ?>
       </ul>
    <?php endif; ?>
    <form method="post" action="index.php">
       <label> Nombre </label>
       <br />
       <input type="text" name="nombre" value="<?php echo $nombre ?>" />
       <br />
       <label> Edad </label>
       <br />
       <input  type="text"  name="edad"  size="3"  value="<?php  echo  $edad
?>" />
       <br />
```

```
        <label> E-mail </label>
        <br />
        <input type="text" name="email" value="<?php echo $email ?>" />
        <br />
        <input type="submit" value="Enviar" />
    </form>
 </body>
</html>
```

Descargar ejemplo

Subir archivos al servidor

Los sitios web más modernos, como por ejemplo las redes sociales están formados en gran parte por material que van subiendo sus usuarios como fotos, música, vídeos, etc. Para ello la aplicación web debe darle al visitante la posibilidad de subir estos archivos. Hasta ahora hemos visto como enviar formularios mediante campos de ingreso de texto o campos que nos permiten seleccionar entre un grupos de opciones. En esta ocasión veremos el elemento que nos está faltando: **input file** y como recibe la información que ha enviado éste en el servidor.

Para realizar este ejemplo vamos a crear un nuevo proyecto en nuestro localhost, yo en mi caso lo llamaré **subir_archivos**.

El mismo tendrá un archivo, **index.php** y una carpeta llamada **archivos**. Ahora vamos a editar el archivo **index.php** con el siguiente código:

```
<!DOCTYPE html>
<html>
 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title> Subir archivos </title>
 </head>
 <body>
    <form method="post" action="index.php" enctype="multipart/form-data">
       <label> Archivo </label>
       <input type="file" name="archivo" required="required" />
       <input type="submit" value="Subir" />
    </form>
 </body>
</html>
```

Bien, un código muy sencillo, sin embargo vamos a analizar dos cosas antes de seguir. En primer lugar usamos el elemento **input file**, el cual nos permite cargar un archivo:

```
<input type="text" name="archivo" required="required" />
```

Además, otra cosa a tener en cuenta es que al formulario le agregamos un atributo **enctype**, con el valor "multipart/form-data". Esto es necesario para que el formulario sea capaz de enviar archivos al servidor:

```
enctype="multipart/form-data"
```

Bien, ahora para recuperar un archivo enviado desde el navegador al servidor debemos hacerlo mediante la variable **$_FILES**, que es un array en donde el índice corresponde al **name** del campo con el valor que se envió, en nuestro caso con el nombre 'archivo'. A su vez cada posición será un array asociativo con los índices:

- **name**: El nombre del archivo con el que se subió al servidor, por ejemplo manzana.jpg.
- **type**: Qué tipo de archivo, por ejemplo si es un .jpg será image/jpeg.
- **tmp_name**: El archivo al ser subido se guardará automáticamente a un directorio temporal, el valor de éste es justamente la ruta al mismo.
- **error**: Nos devolverá un error si se ha producido alguno al intentar recuperar este archivo desde el servidor, de lo contrario devolverá 0.
- **size**: Es el tamaño en bytes del archivo.

Ahora para recuperar el archivo desde el servidor y guardarlo en la carpeta **archivos** haremos lo siguiente:

```
<?php
   $archivo = (isset($_FILES['archivo'])) ? $_FILES['archivo'] : null;
   if ($archivo) {
      $ruta_destino_archivo = "archivos/{$archivo['name']}";
      $archivo_ok       =       move_uploaded_file($archivo['tmp_name'],
$ruta_destino_archivo);
   }
?>
```

Primero guardamos en la variable **$archivo** el resultado si se ha enviado o no un archivo:

```
$archivo = (isset($_FILES['archivo'])) ? $_FILES['archivo'] : null;
```

Luego preguntamos con un **if** el resultado de la variable anterior, osea si hay algún archivo que el usuario ha intentado subir.

De ser así creamos una variable donde guardamos la ruta en donde queremos guardar el archivo:

```
$ruta_destino_archivo = "archivos/{$archivo['name']}";
```

En nuestro caso, dentro de la carpeta **archivos** con el mismo nombre con la que lo envió el usuario.

Y finalmente usamos la función **move_uploaded_file()**, ésta recibirá dos parámetros, la ruta donde se encuentra el archivo actualmente y adónde lo queremos mover:

```
$archivo_ok          =          move_uploaded_file($archivo['tmp_name'],
$ruta_destino_archivo);
```

Osea, la ruta temporal en donde se guarda automáticamente el archivo, y la nueva ruta, la que queremos nosotros.

Esta función **move_uploaded_file()** devolverá **true** o **false**, dependiendo de si se pudo o no mover el archivo, ese resultado lo guardamos en la variable **$archivo_ok**.

El código finalmente quedaría así:

```
<?php
$archivo = (isset($_FILES['archivo'])) ? $_FILES['archivo'] : null;
if ($archivo) {
   $ruta_destino_archivo = "archivos/{$archivo['name']}";
   $archivo_ok         =         move_uploaded_file($archivo['tmp_name'],
$ruta_destino_archivo);
}
?>
<!DOCTYPE html>
<html>
 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title> Subir archivos </title>
 </head>
 <body>
    <?php if (isset($archivo)): ?>
       <?php if ($archivo_ok): ?>
          <strong> El archivo ha sido subido correctamente. </strong>
       <?php else: ?>
          <span style="color: #f00;"> Error al intentar subir el archivo.
</span>
       <?php endif; ?>
    <?php endif; ?>
    <form method="post" action="index.php" enctype="multipart/form-data">
       <label> Archivo </label>
       <input type="file" name="archivo" required="required" />
       <input type="submit" value="Subir" />
    </form>
 </body>
</html>
```

Al código html finalmente agregamos un condicional preguntando si existe un archivo que ha intentado subir el usuario y de ser así si se ha podido subir:

```
<?php if (isset($archivo)): ?>
    <?php if ($archivo_ok): ?>
       <strong> El archivo ha sido subido correctamente. </strong>
    <?php else: ?>
       <span style="color: #f00;"> Error al intentar subir el archivo.
</span>
    <?php endif; ?>
<?php endif; ?>
```

Ahora, una de las cosas más importantes que tenemos que tener en cuenta es que cuando le damos la posibilidad al usuario de subir archivos, debemos tener un filtro de los tipos que éste puede subir. Por ejemplo supongamos que nuestra aplicación sólo permite subir imágenes con extensión **jpg**, **jpeg**, **gif** o **png**, antes de mover el archivo a nuestra carpeta **archivos**, debemos preguntar si éste cumple con las condiciones. Así que vamos a modificar la cabecera:

```
<?php
   $archivo = (isset($_FILES['archivo'])) ? $_FILES['archivo'] : null;
   if ($archivo) {
      $ruta_destino_archivo = "archivos/{$archivo['name']}";
      $archivo_ok        =        move_uploaded_file($archivo['tmp_name'],
$ruta_destino_archivo);
   }
?>
```

Por esto:

```
<?php
   $archivo = (isset($_FILES['archivo'])) ? $_FILES['archivo'] : null;
   if ($archivo) {
      $extension = pathinfo($archivo['name'], PATHINFO_EXTENSION);
      $extension = strtolower($extension);
      $extension_correcta = ($extension == 'jpg' or $extension == 'jpeg'
or $extension == 'gif' or $extension == 'png');
     if ($extension_correcta) {
        $ruta_destino_archivo = "archivos/{$archivo['name']}";
        $archivo_ok       =       move_uploaded_file($archivo['tmp_name'],
$ruta_destino_archivo);
     }
  }
?>
```

Como se ve en el ejemplo mediante las líneas:

```
$extension = pathinfo($archivo['name'], PATHINFO_EXTENSION);
$extension = strtolower($extension);
```

Recuperamos la extensión del archivo mediante a su **name**, osea el nombre del archivo, que será algo como manzana.jpg, y nosotros obtenemos dentro de la variable **$extension** valga la redundancia, la extensión del archivo, por ejemplo **jpg**. Luego pisamos esta variable con el mismo valor, pero todo en minúsculas, esto se debe a que es muy común en Windows que haya archivo con la extensión en mayúscula.

Luego preguntamos si la extensión es correcta y de serlo intentamos mover el archivo:

```
if ($extension_correcta) {
   $ruta_destino_archivo = "archivos/{$archivo['name']}";
   $archivo_ok         =         move_uploaded_file($archivo['tmp_name'],
$ruta_destino_archivo);
}
```

Y finalmente para mostrar por pantalla el resultado del mismo, vamos a modificar:

```
<?php if (isset($archivo)): ?>
    <?php if ($archivo_ok): ?>
       <strong> El archivo ha sido subido correctamente. </strong>
    <?php else: ?>
       <span style="color: #f00;"> Error al intentar subir el archivo.
</span>
    <?php endif; ?>
<?php endif; ?>
```

Por esto:

```
<?php if (isset($archivo)): ?>
   <?php if (!$extension_correcta): ?>
      <span style="color: #f00;"> La extensión es incorrecta, el archivo
debe ser jpg, jpeg, gif o png. </span>
   <?php elseif (!$archivo_ok): ?>
      <span style="color: #f00;"> Error al intentar subir el archivo.
</span>
   <?php else: ?>
      <strong> El archivo ha sido subido correctamente. </strong>
      <br />
      <img src="archivos/<?php echo $archivo['name'] ?>" alt="" />
   <?php endif ?>
 <?php endif; ?>
```

Primero preguntamos si la extensión es incorrecta y de serlo mostramos por pantalla el mensaje con el error:

```
<?php if (!$extension_correcta): ?>
      <span style="color: #f00;"> La extensión es incorrecta, el archivo
debe ser jpg, jpeg, gif o png. </span>
```

Luego si el hubo algún error al intentar subir el archivo:

```
<?php elseif (!$archivo_ok): ?>
      <span style="color: #f00;"> Error al intentar subir el archivo.
</span>
```

Y de que estas dos condiciones no se cumpla ninguna, osea que el archivo tiene una extensión correcta y se subió bien mostramos el mensaje y la imagen que acaba de subir el usuario:

```
<?php else: ?>
   <strong> El archivo ha sido subido correctamente. </strong>
   <br />
   <img src="archivos/<?php echo $archivo['name'] ?>" alt="" />
<?php endif ?>
```

El código entonces finalmente quedaría así:

```
<?php
$archivo = (isset($_FILES['archivo'])) ? $_FILES['archivo'] : null;
if ($archivo) {
   $extension = pathinfo($archivo['name'], PATHINFO_EXTENSION);
   $extension = strtolower($extension);
```

```
   $extension_correcta = ($extension == 'jpg' or $extension == 'jpeg' or
$extension == 'gif' or $extension == 'png');
   if ($extension_correcta) {
      $ruta_destino_archivo = "archivos/{$archivo['name']}";
      $archivo_ok        =        move_uploaded_file($archivo['tmp_name'],
$ruta_destino_archivo);
   }
}
?>
<!DOCTYPE html>
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title> Subir archivos </title>
 </head>
 <body>
    <?php if (isset($archivo)): ?>
       <?php if (!$extension_correcta): ?>
         <span  style="color:  #f00;">  La  extensión  es  incorrecta,  el
archivo debe ser jpg, jpeg, gif o png. </span>
       <?php elseif (!$archivo_ok): ?>
         <span style="color: #f00;"> Error al intentar subir el archivo.
</span>
      <?php else: ?>
         <strong> El archivo ha sido subido correctamente. </strong>
         <br />
         <img src="archivos/<?php echo $archivo['name'] ?>" alt="" />
      <?php endif ?>
   <?php endif; ?>
   <form method="post" action="index.php" enctype="multipart/form-data">
      <label> Archivo </label>
      <input type="file" name="archivo" required="required" />
      <input type="submit" value="Subir" />
   </form>
 </body>
</html>
```

Cookies

Hasta ahora hemos aprendido cómo enviar información de una página a otra por medio de los formularios, y recuperar esos valores mediante las variables $_GET y $_POST. Sin embargo en una aplicación web siempre se necesita mantener información de los visitantes de nuestra página en forma persistente, osea entre página y página, para así tener un control de esta información.

Existen dos formas de lograr esto, mediante las cookies y las sesiones. En esta publicación veremos la primera, cookies, que como decíamos antes permiten guardar la información de los usuarios y así por ejemplo saber si el visitante ya ha ingresado a nuestro sitio o es la primera vez.

Ahora, qué es una cookie, probablemente lo sepas, pero si no es así, simplemente te diré que las cookies son información que se guardan en nuestro navegador. Por tanto el servidor guardará esta información en el cliente, la eliminará o bien, preguntaré si la misma existe.

Para crear una cookie debemos utilizar la función de PHP:

```
setcookie()
```

Esta puede recibir siete parámetros, los dos primeros son obligatorios. El primero es el nombre de la cookie y el segundo el valor. El tercer parámetro es la fecha UNIX hasta donde durará la misma. El cuarto el path en donde estará disponible la cookie. El quinto, el dominio, por ejemeplo mipagina.com. El sexto es si esa cookie debe crearse mediante una conexión segura, osea "https", de ser **true** tomará esta regla. Y el séptimo, significa que ésta sólo será creada mediante el protocolo HTTP, para evitar ataques de scripting como Javascript.

Crear cookie

```
setcookie('nombre', 'Juan', time() + 86400, '/');
```

En este caso guardaremos una cookie que se llamará 'nombre' y tendrá un valor 'Juan', que durará un día, osea sumamos a la función time(), que recordemos nos devolverá la fecha y hora UNIX del momento y le sumamos 86400 (la cantidad de segundos que tiene un día) Y finalmente el path donde estará disponible la cookie, al ser el valor '/', la misma se encontrará en todo el sitio.

Acceder a una cookie

Para acceder a una cookie, por ejemplo la que creamos antes:

```
$_COOKIE['nombre']
```

Eliminar cookie

Para eliminarla, simplemente tenemos que llamar a la función **setcookie()**, como cuando la creamos, pero en lugar de sumar segundos a la fecha UNIX, le restamos a la fecha actual:

```
setcookie('nombre', 'Juan', time() - 86400, '/');
```

Ahora veremos un pequeño ejemplo, para ello crearemos una pequeña aplicación con tres archivos: **index.php**, **guardar_nombre.php** y **eliminar_nombre.php**.

Vamos a editar el código de **index.php** de la siguiente forma:

```
<?php
   $nombre = (isset($_COOKIE['nombre'])) ? $_COOKIE['nombre'] : null;
?>
```

```
<!DOCTYPE html>
<html>
 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title> Página de bienvenida </title>
 </head>
 <body>
    <?php if ($nombre): ?>
       <h1> Bienvenido/a <?php echo $nombre ?> </h1>
       <p> ¿Qué querés hacer? <a href="guardar_nombre.php"> Cambiar mi
nombre </a> | <a href="eliminar_nombre.php"> Eliminar mi nombre </a> </p>
    <?php else: ?>
       <h1> Bienvenido/a usuario desconocido </h1>
       <p>  <a  href="guardar_nombre.php">  Seleccionar  un  nombre  para
identificarme </a> </p>
    <?php endif; ?>
 </body>
</html>
```

En primer lugar preguntamos si existe una cookie llamada 'nombre', y de serlo así guardamos el valor en una variable, pero sino guardará **null**:

```
<?php
   $nombre = (isset($_COOKIE['nombre'])) ? $_COOKIE['nombre'] : null;
?>
```

Y luego preguntaremos si existe un nombre, si es así saludamos al visitante por ese nombre y mostramos un link para cambiar el nombre o bien para eliminarlo. De no ser así saludamos al usuario, como "usuario desconocido" y le mostramos un link para que guarde su nombre:

```
<?php if ($nombre): ?>
   <h1> Bienvenido/a <?php echo $nombre ?> </h1>
   <p> ¿Qué querés hacer? <a href="guardar_nombre.php"> Cambiar mi nombre
</a> | <a href="eliminar_nombre.php"> Eliminar mi nombre </a> </p>
<?php else: ?>
   <h1> Bienvenido/a usuario desconocido </h1>
   <p>   <a   href="guardar_nombre.php">   Seleccionar   un   nombre   para
identificarme </a> </p>
<?php endif; ?>
```

Ahora editaremos el archivo **guardar_nombre.php** con el siguiente código:

```
<?php
 $nombre = (isset($_COOKIE['nombre'])) ? $_COOKIE['nombre'] : null;
 if($_SERVER['REQUEST_METHOD'] == 'POST'){
    $nombre = $_POST['nombre'];
    setcookie('nombre', $nombre, time() + 86400, '/');
    header('Location: index.php');
    exit;
 }
?>
<!DOCTYPE html>
<html>
 <head>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title> Guardar nombre </title>
 </head>
 <body>
    <form method="post" action="guardar_nombre.php">
       <label> Nombre: </label>
       <input type="text" name="nombre" value="<?php echo $nombre ?>"
required="required" />
       <input type="submit" value="Guardar" />
       <a href="index.php"> Volver </a>
    </form>
 </body>
</html>
```

Aquí mostramos un formulario con un campo de entrada para ingresar el nombre y un botón para guardarlo:

```
<form method="post" action="guardar_nombre.php">
   <label> Nombre: </label>
   <input type="text" name="nombre" value="<?php echo $nombre ?>"
required="required" />
   <input type="submit" value="Guardar" />
   <a href="index.php"> Volver </a>
</form>
```

En la cabecera preguntamos si existe ese cookie, para luego mostrarlo como valor por defecto del campo de entrada:

```
$nombre = (isset($_COOKIE['nombre'])) ? $_COOKIE['nombre'] : null;
```

Y finalmente preguntaremos si existe una petición por POST, osea si se ha enviado el formulario y guardamos la cookie con el nombre que ha enviado el usuario y redireccionamos a la página de bienvenida, para así mostrar ese nombre.

```
if($_SERVER['REQUEST_METHOD'] == 'POST'){
   $nombre = $_POST['nombre'];
   setcookie('nombre', $nombre, time() + 86400, '/');
   header('Location: index.php');
   exit;
}
```

Ahora crearemos la acción para eliminar la cookie editando el archivo **eliminar_nombre.php** con el siguiente código:

```
<?php
 if(isset($_COOKIE['nombre'])){
    setcookie('nombre', '', time() - 86400, '/');
    header('Location: index.php');
    exit;
 }
?>
```

Aquí simplemente eliminamos la cookie, seteándola como hicimos cuando la creamos, pero en lugar de sumar un día, lo restamos, para así forzar su vencimiento.

Descargar ejemplo

Sesiones

Continuando con la publicación pasada en donde habíamos aprendido cookies, una de las formas de guardar información del visitante de nuestro sitio para que ésta esté disponible entre página y página. En esta ocasión aprenderemos otra de las formas: sesiones.

Cookies Vs Sesiones

Habíamos dicho que ambas tienen un mismo fin: guardar información del visitante entre página y página, sin embargo hay algo que diferencia a ambas formas que hay que tener muy presente a la hora de crear nuestra aplicación web.

Tanto los cookies como las sesiones, son una especia de comunicación, una identificación que hay entre el navegador del visitante y el servidor web. En la publicación pasada guardábamos el nombre del visitante, por ejemplo "Juan", sin embargo ese valor se está guardando en el navegador del usuario, eso significa que un usuario de internet con conocimientos medios puede modificar ese valor, ya que está en su propio disco duro.

Pensemos en algo un poco más complejo que guardar un simple nombre: un login. El usuario ingresa su nombre y su contraseña, se buscan estos datos en una base de datos y se guarda un identificador, un número. Si ese número, que corresponde a ese usuario se guarda en una cookie, el visitante puede modificar esa cookie, y hacerse pasar por otro usuario. Sin embargo con una sesión, eso no pasaría, ya que los valores, por ejemplo ese identificador se guardarán en el servidor y no en el navegador.

Iniciar sesión

Cuando trabajemos con sesiones en nuestro script debemos incluir la siguiente función:

```
session_start();
```

Si no existe una sesión actualmente creará una, de lo contrario continuará con la sesión abierta hasta el momento.

Guardar valores en sesión

Las sesiones, al igual que las cookies son un array asociativo. Por ejemplo para guardar el nombre de usuario dentro de una sesión deberíamos hacer lo siguiente:

```
$_SESSION['nombre'] = 'fernando';
```

Y obviamente para acceder al mismo llamar a ese array **$_SESSION** con el índice correspondiente:

```
$_SESSION['nombre'];
```

Eliminar valores de sesión

Siguiendo con el ejemplo 'nombre', si quisiéramos eliminarlo deberíamos utilizar la función **unset()** que lo que hará es destruir variables e índices de arrays, incluso los de una sesión:

```
unset($_SESSION['nombre']);
```

Finalizar sesión

Si en cambio lo que queremos hacer es eliminar absolutamente la sesión, debemos usar lo siguiente:

```
session_destroy();
```

Sin embargo, yo aconsejo no usar esta función, ya que romperá la sesión y con ella todos sus índice y valores. Nosotros independientemente de si el usuario está logueado o no, podemos llevar un control para saber por ejemplo cuántas veces ha iniciado sesión y a qué página ha ingresado.

Para entender mejor cómo funciona esto de las sesiones vamos a crear un pequeño proyecto con tres archivos a los que yo voy a llamar: **index.php**, **usuario.php** y **cerrar_sesion.php**.

Dentro de **index.php** vamos a escribir el siguiente código:

```
<?php
 session_start();
 $usuarios = array(
    array('nombre' => 'roberto', 'contrasena' => '1234'),
    array('nombre' => 'jorge', 'contrasena' => '1234'),
    array('nombre' => 'toni', 'contrasena' => '1234')
 );
 if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $nombre = $_POST['nombre'];
    $contrasena = $_POST['contrasena'];
    //Creamos una variable para verificar si el usuario con ese nombre y
contraseña existe.
    $usuario_encontrado = false;
    foreach($usuarios as $item){
       //Si encuentra al usuario con ese nombre y contraseña sete la
variable $usuario_encontrado a true y rompe el bucle para no seguir
buscando.
       if($nombre    ==    $item['nombre']    and    $contrasena    ==
$item['contrasena']){
          $usuario_encontrado = true;
          break;
```

```
        }
    }
    //Verifica si dentro del bucle se ha encontrado el usuario.
    if($usuario_encontrado){
        $_SESSION['logueado'] = true;
        $_SESSION['nombre'] = $nombre;
        header('Location: usuario.php');
        exit;
    }else{
        $error_login = true;
    }
 }
?>
<!DOCTYPE html>
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title> Logueo </title>
 </head>
 <body>
    <?php if(isset($error_login)): ?>
        <span  style="color:  #f00;">  El  usuario  o  la  contraseña  son
incorrectos. </span>
    <?php endif; ?>
    <form method="post" action="index.php">
        <label for="nombre"> Nombre </label>
        <input  type="text"  name="nombre"  id="nombre"  required="required"
/>
        <label for="contrasena"> Contraseña </label>
        <input    type="password"    name="contrasena"    id="contrasena"
required="required" />
        <input type="submit" value="Enviar" />
    </form>
 </body>
</html>
```

En primer lugar creamos un formulario para ingresar un nombre de usuario y la contraseña, nada nuevo:

```
<form method="post" action="index.php">
   <label for="nombre"> Nombre </label>
   <input type="text" name="nombre" id="nombre" required="required" />
   <label for="contrasena"> Contraseña </label>
   <input      type="password"      name="contrasena"      id="contrasena"
required="required" />
   <input type="submit" value="Enviar" />
</form>
```

Luego, en la cabecera verificamos si ha llegado una petición por POST, de la acción al recibir el formulario:

```
<?php
 session_start();
 $usuarios = array(
    array('nombre' => 'roberto', 'contrasena' => '1234'),
    array('nombre' => 'jorge', 'contrasena' => '1234'),
```

```
    array('nombre' => 'toni', 'contrasena' => '1234')
 );
 if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $nombre = $_POST['nombre'];
    $contrasena = $_POST['contrasena'];
    //Creamos una variable para verificar si el usuario con ese nombre y
contraseña existe.
    $usuario_encontrado = false;
    foreach($usuarios as $item){
       //Si encuentra al usuario con ese nombre y contraseña sete la
variable $usuario_encontrado a true y rompe el bucle para no seguir
buscando.
       if($nombre     ==     $item['nombre']     and     $contrasena     ==
$item['contrasena']){
          $usuario_encontrado = true;
          break;
       }
    }
    //Verifica si dentro del bucle se ha encontrado el usuario.
    if($usuario_encontrado){
       $_SESSION['logueado'] = true;
       $_SESSION['nombre'] = $nombre;
       header('Location: usuario.php');
       exit;
    }else{
       $error_login = true;
    }
 }
?>
```

Bueno, como dijimos antes, cuando trabajamos con sesiones debemos llamar dentro de nuestro script a la función **session_start()**:

```
session_start();
```

Y también tendremos un array con una lista de usuarios, esto se debe a que todavía no hemos trabajado con base de datos, así que vamos a simular una:

```
$usuarios = array(
    array('nombre' => 'roberto', 'contrasena' => '1234'),
    array('nombre' => 'jorge', 'contrasena' => '1234'),
    array('nombre' => 'toni', 'contrasena' => '1234')
);
```

Bien, ahora vamos a analizar el código dentro del **if** que verifica si se ha enviado el formulario:

Primero guardamos los datos de usuario y contraseña:

```
$nombre = $_POST['nombre'];
$contrasena = $_POST['contrasena'];
```

Creamos una variable **$usuario_encontrado** que guardaré el resultado de si existe algún usuario con ese nombre y contraseña, por defecto será **false**.

```
$usuario_encontrado = false;
```

Luego con un **foreach** recorremos la lista de usuarios:

```
foreach($usuarios as $item){
   //Si encuentra al usuario con ese nombre y contraseña sete la variable
$usuario_encontrado a true y rompe el bucle para no seguir buscando.
   if($nombre == $item['nombre'] and $contrasena == $item['contrasena']){
      $usuario_encontrado = true;
      break;
   }
}
```

En cada loop preguntamos si el nombre de usuario y la contraseña coinciden con la posición en que estamos parados y de ser así seteamos la variable **$usuario_encontrado** a **true** y rompemos el ciclo de repetición, ya no es necesario seguir comprobando:

```
if($nombre == $item['nombre'] and $contrasena == $item['contrasena']){
   $usuario_encontrado = true;
   break;
}
```

Finalmente con un **if** verificamos si el usuario ha sido encontrado con ese nombre y contraseña:

```
if($usuario_encontrado){
   $_SESSION['logueado'] = true;
   $_SESSION['nombre'] = $nombre;
   header('Location: usuario.php');
   exit;
}else{
   $error_login = true;
}
```

De ser así, entonces seteamos dos índices en nuestra sesión, 'logueado' a **true**, para comprobar en el resto de nuestra página si el usuario ha iniciado una sesión correctamente, y 'nombre' con el nombre del usuario, valga la redundancia. Además lo redireccionamos a una página principal para usuarios, que editaremos en un momento:

```
$_SESSION['logueado'] = true;
$_SESSION['nombre'] = $nombre;
header('Location: usuario.php');
exit;
```

Pero si el nombre de usuario y la contraseña están mal, entonces creamos una variable **$error_login**:

```
$error_login = true;
```

Para luego mostrar el mensaje de error dentro del código html:

```
 <?php if(isset($error_login)): ?>
```

```
    <span style="color: #f00;"> El usuario o la contraseña son
incorrectos. </span>
 <?php endif; ?>
```

Bien, con esto ya tenemos nuestro login, en la página principal.

Ahora vamos a editar nuestro archivo **usuario.php** con el siguiente código:

```
<?php
 session_start();
 if(isset($_SESSION['logueado']) and $_SESSION['logueado']){
    $nombre = $_SESSION['nombre'];
 }else{
    //Si el usuario no está logueado redireccionamos al login.
    header('Location: index.php');
    exit;
 }
?>
<!DOCTYPE html>
<html>
 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title> Página de bienvenida </title>
 </head>
 <body>
    <h1> Bienvenido/a <?php echo $nombre; ?> </h1>
    <p> <a href="cerrar_sesion.php"> Cerrar sesión </a> </p>
 </body>
</html>
```

Aquí simplemente mostramos un mensaje saludando al usuario, que se ha logueado:

```
<h1> Bienvenido/a <?php echo $nombre; ?> </h1>
```

Y también un link para cerrar sesión:

```
<p> <a href="cerrar_sesion.php"> Cerrar sesión </a> </p>
```

En la cabecera vamos a comprobar también que para ingresar a esta página el usuario se haya logueado previamente.

```
if(isset($_SESSION['logueado']) and $_SESSION['logueado']){
   $nombre = $_SESSION['nombre'];
}else{
   //Si el usuario no está logueado redireccionamos al login.
   header('Location: index.php');
   exit;
}
```

Si el usuario está logueado guardamos el nombre en una variable **$nombre** para luego saludarlo, pero de no ser así, lo redireccionamos a la página principal, ya que no está autorizado a ver esa página.

Y finalmente editamos el código de **cerrar_sesion.php**:

```
<?php
 session_start();
 if (isset($_SESSION['logueado']) and $_SESSION['logueado']) {
    $_SESSION['logueado'] = false;
    unset($_SESSION['nombre']);
 }
 header('Location: index.php');
 exit;
?>
```

Aquí simplemente preguntamos si el usuario está logueado, lo deslogueamos seteando la variable 'logueado' a **false** y eliminando el nombre de memoria:

```
$_SESSION['logueado'] = false;
unset($_SESSION['nombre']);
```

Bueno, con esto terminamos el trabajo de cookies y sesiones.

Saludos!

Descargar ejemplo

Introducción a MySQL, tablas

Hasta ahora hemos visto en varias ocasiones la necesidad de trabajar con datos, con información, que al carecer de una base de datos hemos simulado mediante otro tipo de información como arrays. Sin embargo una aplicación web moderna, ya sea un foro, un blog, un sitio de compras online, se alimenta de información extraída de una base de datos.

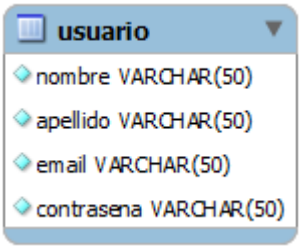
Si nunca has trabajado con base de datos, no deberías preocuparte porque todo lo que pienso publicar a partir de ahora, estará orientado a principiantes. Pero si en cambio tenés amplios conocimientos de base de datos tal vez te aburras un poco al principio, podés saltear las siguientes publicaciones hasta retomar con PHP.

MySQL es un tipo de base de datos. Así como existen distintos lenguajes de programación como PHP, Python, Java, Ruby, etc; también existen distintas bases de datos, ya se MySQL, Oracle, Postgresql, Sqlite, etc. La razón por la cual decidí utilizar MySQL, es porque si bien PHP puede conectarse a distintas bases de datos, MySQL es con la que mejor relación tiene, y con la que se suelen utilizar más los desarrollos en este lenguaje.

Tablas

Una base de datos SQL como MySQL, está formado por tablas. Estas tablas nos permiten almacenar información, dependiendo de qué queramos guardar, tendremos una tabla para cada caso. Con esto me refiero a que yo puede tener una tabla para guardar usuarios, una para guardar países, comentarios, ventas y cualquier cosa que se te pueda ocurrir.

A su vez estas tablas tendrán atributos, que serán los distintos valores que tendrá cada registro. Por ejemplo yo puedo crear una tabla para guardar usuarios con cuatro atributos; nombre, apellido, email y contraseña, dependiendo obviamente de los tipos de datos que sean precisos guardar, de lo que necesite la aplicación en sí.



Como se ve en el ejemplo tenemos una tabla de nombre **usuario**, que tiene cuatro atributos: **nombre**, **apellido**, **email** y **contrasena**. Con respecto a la palabra **VARCHAR(50)**, es el tipo de datos, significa que será una cadena y soportará hasta 50 caracteres, pero más adelante veremos tipos de datos. La razón de por qué el atributo es 'contrasena' y no 'contraseña' se debe a que MySQL sólo soporta caracteres alfanuméricos y guión bajo en los nombres de sus atributos, el caracter especial 'ñ' no está permitido.

Entonces teniendo nuestra primer tabla, podemos simular que empezamos a insertar registros dentro de la misma:

nombre	**apellido**	**email**	**contrasena**
Juan	Pérez	juan@mail.com	1234
Susana	Giménez	susana@mail.com	1234
Ricardo	Caruso Lombardi	caruso@mail.com	1234
Mariana	Rodríguez	mariana@mail.com	1234

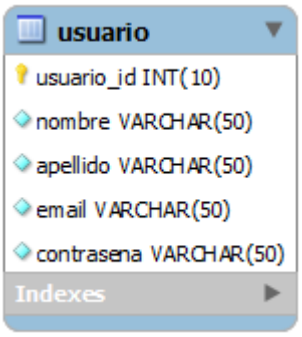
Como se ve en el ejemplo, hemos guardado una pequeña lista con cuatro usuarios, cada uno de estos se llamará registro. Con respecto a los valores, obviamente será acordes a lo que corresponde cada uno: un nombre, un apellido, una dirección de email y la contraseña. En realidad es una base de datos, en las tablas que guardan usuarios, las contraseña no suelen ser visible la cadena original, ya que estás pasan primero por un proceso de encriptado o hasheo, pero eso lo veremos más adelante.

Clave primaria

En el ejemplo anterior nosotros simplemente teníamos una tabla con cuatro registros, con cuatro usuarios. El asunto ahora es que cada uno de esos registros, luego de ser insertados serán servido a nuestra aplicación, para realizar distintas operaciones. Para recuperar sus valores, para modificarlos, o incluso para eliminarlos. Así que cada uno de esos registros debería tener un identificador, algo que lo diferencie del resto, para poder acceder fácilmente al mismo.

Por ejemplo, el primer registro se trata de un usuario llamado 'Juan Pérez', que teniendo en cuenta que en nuestro idioma castellano 'Juan Pérez' es un nombre y apellido muy común, seguramente tendremos otros usuarios con el mismo nombre y apellido, entonces ¿Cómo saber cuál es el que estamos buscando? Otra alternativa podría ser su dirección de email, ya que por lógica ésta debería ser única e irrepetible, el problema es que si este usuario modifica su email le perderíamos el rastro. Deberíamos tener algo que lo identifique desde que el registro es insertado en la tabla. Aquí entra en juego lo que se conoce como **clave primaria** o **primary key**.

Una clave primaria, es el identificador, el id de un registro. Este dato es único e irrepetible, y por lógica no debería ser modificado nunca.



Ahora que nuestra tabla ha sido modificada podemos pasar a mostrar cómo quedarían los registros que antes simulamos, con el nuevo cambio:

usuario_id	nombre	apellido	email	contrasena
1	Juan	Pérez	juan@mail.com	1234
2	Susana	Giménez	susana@mail.com	1234
3	Ricardo	Caruso Lombardi	caruso@mail.com	1234
4	Mariana	Rodríguez	mariana@mail.com	1234

De este modo nuestra aplicación tendrá una forma de referirse a un registro específico, por su id, su clave primaria. No importa cómo se llame el usuario, si Juan Pérez, o cualquier otro nombre, ese registro será el registro 1 de la tabla **usuario**, sin más.

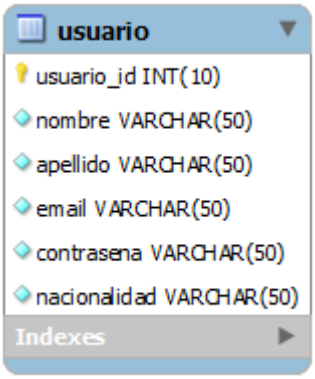
Saludos!

Introducción a MySQL, tablas y relaciones

Habíamos dicho que una base de datos MySQL está formada por tablas, tablas que nos permiten separar por grupos y sus registros. Pero a su vez estas tablas pueden tener

relaciones entre sí. Esto nos permite evitar la redundancia separando cada registro en la tabla a la cual le pertenece.

Vamos a suponer que a nuestra tabla **usuario**, debemos agregarle un nuevo atributo para guardar la nacionalidad del usuario.

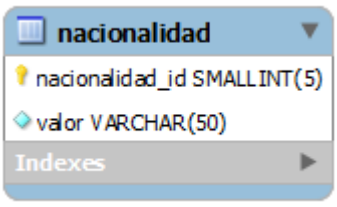


Bien, si seguiste la publicación anterior, verás que no hay nada nuevo en esto, así que simplemente podemos comenzar a simular la inserción de registros:

usuario_id	nombre	apellido	email	contrasena	nacionalidad
1	Ricardo	Caruso Lombardi	caruso@mail.com	1234	Argentina
2	Susana	Giménez	susana@mail.com	1234	Argentina
3	Wilson	Sanguinetti	wilson@mail.com	1234	Uruguay
4	Ronaldo	Santos	ronaldo@mail.com	1234	Brasil
5	Marisa	Santa Cruz	marisa@mail.com	1234	Paraguay

La nacionalidad es un atributo, que si bien no está mal que esté en la tabla **usuario**, ya que nosotros necesitamos que al guardar cada usuario también guardemos su nacionalidad, sin embargo el guardar la cadena entera: 'Uruguay' por ejemplo, no es una buena idea, es redundante y difícil de mantener en el tiempo.

Porque en nuestra base de datos habrá cientos, miles de personas de nacionalidad uruguaya, argentina, brasileña, paraguaya o de otros países. Por tanto deberíamos crear una tabla sólo para guardar las nacionalidades:

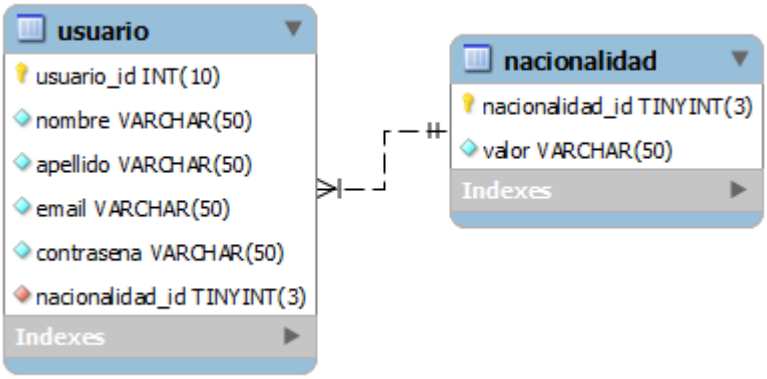


Y aquí simplemente guardaremos los registros con todas las nacionalidades posibles:

nacionalidad_id	valor
1	Argentina
2	Brasil
3	Paraguay
4	Uruguay

Claves externas

Como se ve en el ejemplo anterior, tenemos una tabla **usuario** y una **nacionalidad**, y a su vez dijimos que cada usuario tiene una nacionalidad, por tanto estas tablas deberían estar unidas. Así que para ello vamos a modificar nuestra tabla usuario:



De esta manera la tabla **usuario** y la tabla **nacionalidad** estarán relacionadas entre sí. En el caso de la tabla **usuario**, ésta tendrá un atributo **nacionalidad_id**, a esto se le llama clave externa, o clave foránea. La clave externa **nacionalidad_id** de la tabla **usuario**, coincidirá con la clave primaria **nacionalidad_id** de la tabla **nacionalidad**.

Entonces al insertar registros en la tabla usuario, tendremos que modificar lo que hicimos antes por esto:

usuario_id	nombre	apellido	email	contrasena	nacionalidad_id
1	Ricardo	Caruso Lombardi	caruso@mail.com	1234	1
2	Susana	Giménez	susana@mail.com	1234	1
3	Wilson	Sanguinetti	wilson@mail.com	1234	4
4	Ronaldo	Santos	ronaldo@mail.com	1234	2
5	Marisa	Santa Cruz	marisa@mail.com	1234	3

De este modo nosotros unimos el registro del usuario 1, Ricardo Caruso Lombardi de la tabla **usuario**, con el registro 1 de la tabla **nacionalidad**.

Y así nos evitamos la redundancia, no necesitamos escribir la palabra ‘Argentina’ cada vez que un usuario sea de ese país. Y si la nacionalidad cambia de nombre, simplemente tendremos que cambiar el registro en la tabla principal **nacionalidad**, y no cien mil veces en la tabla **usuario**.

Tipos de relaciones

Ahora que sabemos que dos tablas pueden estar relacionadas, podemos aprender cuáles son los tipos de relación que pueden tener éstas.

Relación de 1 a 1 (uno a uno)

Éstas se dan cuando un registro está relacionado con otro y con ese solamente, y a su vez este segundo también estará relacionado solamente con el primero.

Un ejemplo de esto puede ser un usuario y su teléfono. Es muy común en aplicaciones actuales, con el avance de la telefonía celular, que un usuario pueda registrar su número de celular, por si pierde su contraseña poder recuperarla recibiendo un mensaje de texto. En este caso cada usuario podrá tener un número de teléfono y éste número sólo le va a pertenecer a este usuario.

Relación de 1 a n (uno a muchos)

En este caso un registro estará relacionado sólo con otro, pero este segundo podrá estar relacionado con más de uno.

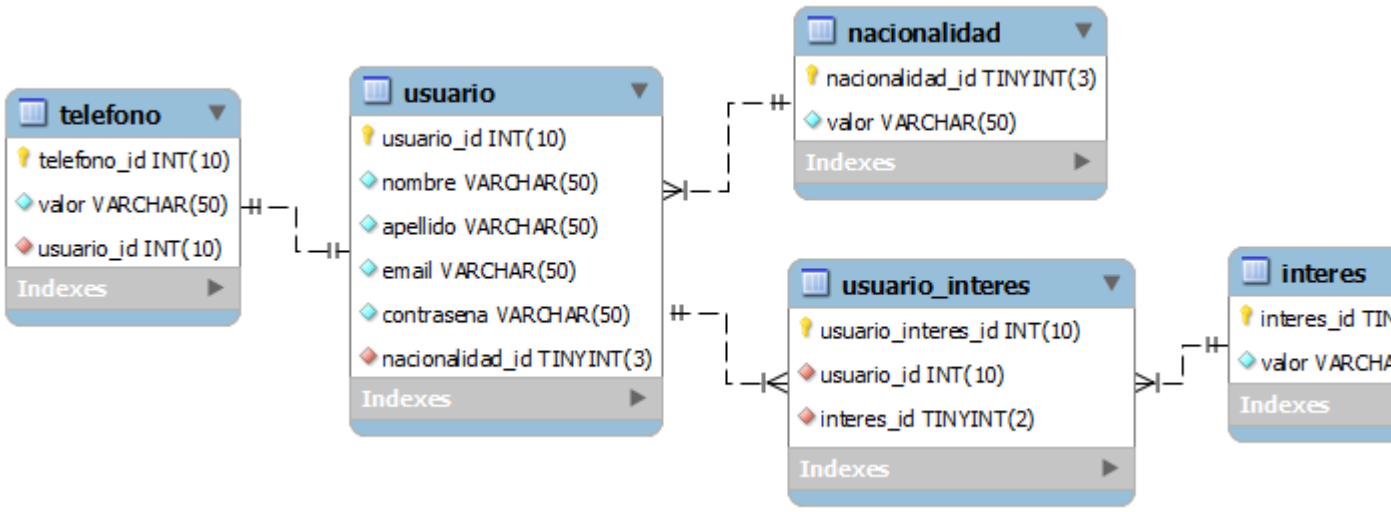
El ejemplo para esto es la relación de tablas que vimos antes, un usuario tendrá una única nacionalidad, pero esta nacionalidad podrá estar en más de un usuario.

Relación de n a n (muchos a muchos)

Aquí un registro tendrá relación con muchos registros, y a su vez cada uno de estos también estará relacionado con varios. Estas relaciones generan siempre una tabla intermedia.

Por ejemplo, nosotros podríamos tener una tabla para guardar usuarios y otra para guardar intereses (música, cine, deportes, etc) entonces, un usuario puede tener varios intereses, y cada uno de estos intereses puede estar presente en varios de estos usuarios. A juan le puede gustar el cine y los deportes, pero a su vez el cine también puede gustarle a Pedro.

Para ello tendremos que crear una tabla con dos claves externas, una para guardar el identificador del usuario y otra para el identificador del interés, y obviamente la clave primaria que identificará la unión entre ambas tablas.



Como se ve en el ejemplo la tabla **usuario** está relacionada con la tabla **telefono**, de 1 a 1, cada usuario tendrá un teléfono, y cada uno de estos teléfonos le va pertenecer a un único usuario.

Además la tabla **usuario** tendrá una relación de 1 a n con **nacionalidad**. Como dijimos antes, cada usuario tendrá una nacionalidad, pero éstas podrán esta presentes en más de un usuario.

Y por último la tabla **usuario** tendrá una relación de n a n con **interes**, lo que generará una nueva tabla llamada **usuario_interes**, para guardar registros que unirán ambas tablas.

Introducción a MySQL, creación de tablas

Continuando con el diagrama de nuestra base de datos, ahora podemos pasar a la acción, creando nuestra primer base de datos y sus tablas. Pero para esto esto primero vamos a detenernos en algo muy importante, las tablas como ya sabemos tienen atributos, y estos atributos tienen un tipo de dato, dependiendo de si son cadenas de texto, números o fechas.

En esta ocasión no voy a mostrar todos los tipos de datos que existen, pero sí los que a mí parecer, son los más usados:

Tipos de datos numéricos

Existen cinco tipos de datos numéricos, estos dependerán del valor máximo que necesitemos guardar, del volumen de información. Por ejemplo si tuviésemos un blog, no será el mismo tipo de dato el que usemos para guardar la cantidad de categorías, que no deberían ser muchas; 5, 10, 15; que la cantidad de comentarios que puede tener ese blog, que pueden llegar a ser miles.

En primer lugar, cada tipo de dato soportará hasta un valor máximo dependiendo de si tiene signo o no. O sea, si soporta valores negativos o sólo positivos. Por ejemplo, el tipo de dato **tinyint**, configurado sin signo (sólo números positivos) tendrá un rango que irá desde el valor **0** hasta el **255**, de lo contrarió será desde **-128** hasta **127**. Este tipo de dato nos podría ser útil para guardar por ejemplo las categorías de un foro que nombramos antes, con 255 sería más que suficiente, pero no para guardar las publicaciones o comentarios de ese blog, ya que puede haber muchísimo más que 255.

Los valores numéricos son:

Tipo	**Bytes**	**Valor Mínimo**	**Valor Máximo**
		(Con signo/Sin signo)	**(Con signo/Sin signo)**
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

(Fuente: https://dev.mysql.com/doc/refman/5.0/es/numeric-types.html)

Tipos de datos de cadenas de caracteres

En MySQL existen cuatro grupos de tipos de datos para guardar cadenas.

Por un lado tenemos **char** y **varchar**, los cuales nos permitirán guardar cadenas de hasta **255** caracteres. La principal diferencia entre ambos es que **char** usa la cantidad de caracteres que asignemos, rellenando los caracteres restantes con espacios. Por ejemplo podemos configurar nuestro atributo como un **char** de 30 caracteres, pero si insertamos registros con valores menores a 30, por ejemplo un valor de 12, éste se guardará con los caracteres que ingresamos, y automáticamente agregará espacios en blanco hasta completar los 30. Esto con **varchar** no sucede, sólo se usarán la cantidad de caracteres del registro guardado. Por esto se suele tener preferencia por **varchar**, ya que nos permite ahorrar memoria comparado con **char**.

char y **varchar** son útiles para guardar cadenas pequeñas, por ejemplo nombres de personas. Pero si necesitamos guardar valores muy grandes, deberíamos usar **blog** o **text**, que nos permitirán almacenar extensas cadenas como por ejemplo la letra de una canción. La diferencia entre ambas es que **blog** almacena datos binarios, mientras que **text** nos

permiten guardar cadenas de caracteres. Por este motivo, **text** es más utilizado que **blog**, ya que es más fácil de trabajar con sus valores.

También tenemos los tipos de datos **binary** y **varbinary**, que son igual a **char** y **varchar**, pero a diferencia de estos, permiten guardar datos binarios y no cadenas de texto.

Y por últimos tendremos los tipos de datos **enum** y **set**, que nos permitirán guardar un valor extraído de una lista. Por ejemplo un atributo **sexo**, que sólo podrá ser dos valores: 'hombre' o 'mujer'. La diferencia entre ambos es que **enum** sólo guardará un valor de la lista especificada, mientras que **set** puede guardar varios de estos separado por coma, por ejemplo: 'hombre,mujer'.

Tipos de datos de fecha y hora

Los tipos de dato de fecha y hora son muy útiles a la hora de hacer seguimientos, para guardar la fecha de un suceso o la hora del mismo, o incluso ambos.

Estos tipos de datos pueden ser **date**, para guardar una fecha, el formato, por ejemplo para el 23 de Diciembre del año 2013, debe ser de la siguiente manera: '2013-12-23'. Para guardar la hora deberíamos usar el tipo de dato **time**, el formato es algo como esto: '00:56:41′, separado en 'horas:minutos:segundos'.

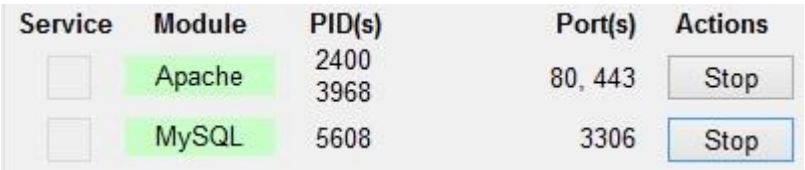
Si en cambio quisiéramos guardar la fecha y hora en un mismo campo deberíamos usar **datetime** o **timestamp**. La diferencia entre ambos es que **datetime** guarda sus valores con el siguiente formato: '2013-12-23 00:56:41', mientras que **timestamp** comprenderá el formato unix, osea la cantidad de segundos desde 1 de Enero de 1970 a las 00:00:00, la misma fecha y hora que guardamos con **datetime** podríamos guardarla en una columna **timestamp** de esta forma '1387756601'. Otra diferencia entre ambos es que si bien **datetime** tiene un formato más amigable ante los ojos humanos, **timestamp** ocupa menos memoria que el primero.

Para profundizar sobre tipos de datos, podés consultar el manual de MySQL:

https://dev.mysql.com/doc/refman/5.0/es/column-types.html

Crear base de datos

Para crear nuestra base de datos, primero debemos asegurarnos que en nuestro localhost esté encendido el servicio MySQL. Así que vamos a abrir nuestro panel de control del xampp, y además de iniciar el servicio Apache, como siempre hacemos, también vamos a pulsar 'Start' en la segunda opción, la de MySQL.



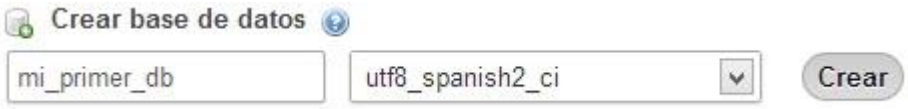
Ambos servicios, tanto Apache como MySQL, deben estar encendidos.

Ahora vamos a abrir nuestro navegador y vamos a ingresar la siguiente dirección:

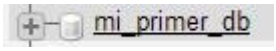
```
https://localhost/phpmyadmin
```

Aquí tendremos un panel con nuestras base de datos, así que vamos a pulsar en el botón 'Bases de datos'

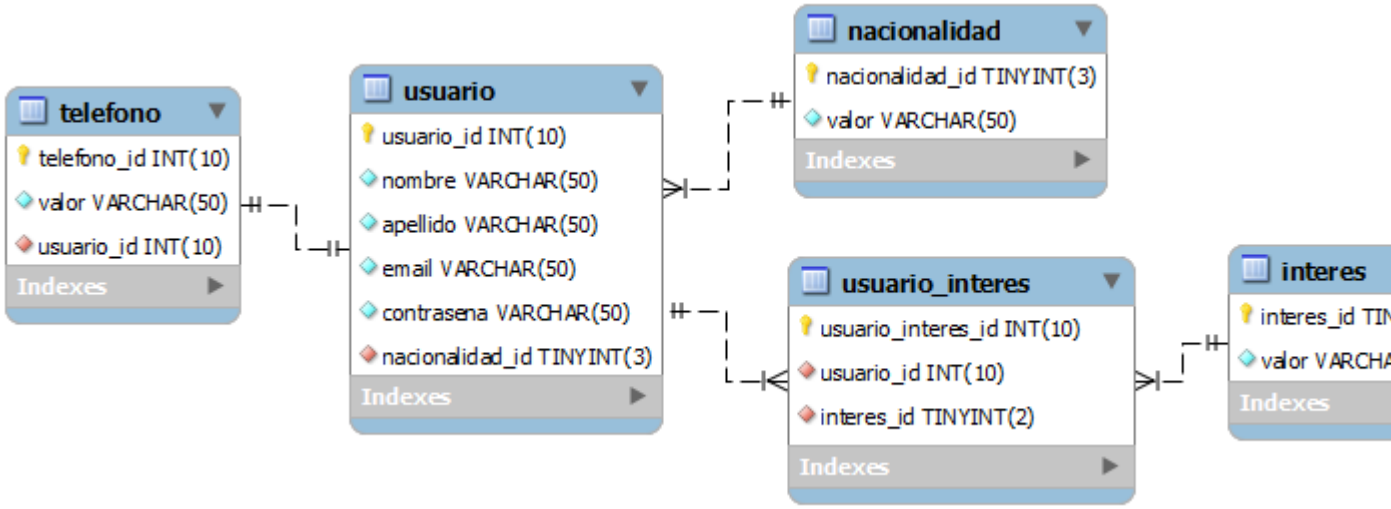
Aquí debemos ingresar un nombre para nuestra base de datos y el cotejamiento, en este último vamos a seleccionar el valor: 'utf8_spanish2_ci', para que nuestras tablas soporten cadenas con caracteres especiales como tildes.



Y acto seguido vamos a seleccionar, en la lista de la izquierda, la base de datos que acabamos de crear



Bien, ya tenemos nuestra base de datos, ahora sólo nos resta crear nuestras tablas, basadas en el diagrama que habíamos creado la última vez.



Cada vez que tengamos que ejecutar código SQL, debemos pulsar el botón, valga la redundancia: 'SQL'

Empecemos por crear la tabla **nacionalidad**, pulsamos 'SQL' y escribimos en el campo de entrada el siguiente código:

```
CREATE TABLE nacionalidad (
   nacionalidad_id tinyint(3) unsigned not null auto_increment primary
key,
   valor varchar(50) not null
)engine=innoDB;
```

Una vez ingresado el código pulsamos el botón 'Continuar' para crear nuestra tabla.

Ahora vamos a analizar el código.

En primer lugar escribimos:

```
CREATE TABLE nombre_de_la_tabla()engine=innoDB;
```

Esto es muy fácil de entender. Creamos una tabla con un nombre X, dentro de la apertura y cierre de paréntesis vamos a definir los atributos de esta tabla. Por último al finalizar con la creación de la tabla definimos el motor de la tabla, que en nuestro caso será **innoDB**.

Los atributos serán sólo dos, la clave primaria y el valor, osea el nombre de la nacionalidad.

```
nacionalidad_id tinyint(3) unsigned not null auto_increment primary key,
```

Lo primero que debemos hacer es definir el nombre del atributo y luego su tipo, entre paréntesis indicamos la cantidad de dígitos que tendrá éste. En este caso definimos **tinyint**, que sin signo soporta hasta 255 caracteres, lo que es más que suficiente para guardar la cantidad de nacionalidades que necesitamos. Como **tinyint** es un valor numérico especificamos que es **unsigned** (en criollo sin signo, sin valores negativos)

Finalmente ingresamos tres instrucciones más para este atributo. Que no sea null (**not null)**, **auto_increment** y **primary key** (clave primaria o identificador) Estos dos últimos valores en conjunto nos permitirán que cuando insertemos un nuevo registro en esta tabla, no sea necesarios setear la clave primaria, por defecto comenzará desde 1, seguirá en 2, 3, 4, 5 y así sucesivamente.

```
valor varchar(50) not null
```

Y por último ingresamos el atributo 'valor' que será de tipo de dato **varchar** de 50 caracteres y **not null**. Con 50 caracteres como máximo es suficiente.

Además como éste es el último atributo para esta tabla, no debemos ingresar coma al final.

Bien, ahora para continuar vamos a crear nuestra tabla **usuario**, pero antes de continuar vamos a retomar lo que vimos en la última publicación, la relación entre tablas. Recordar que la tabla **usuario** y **nacionalidad** están unidas, en una relación de 1 a n (uno a muchos) Entonces vamos a crear la tabla de la siguiente forma:

```
CREATE TABLE usuario (
```

```
   usuario_id int(10) unsigned not null auto_increment primary key,
   nombre varchar(50) not null,
   apellido varchar(50) not null,
   email varchar(50) not null unique,
   contrasena varchar(50) not null,
   fecha_alta datetime not null,
   fecha_modificacion datetime not null,
   nacionalidad_id tinyint(3) unsigned not null,
   foreign key(nacionalidad_id) references nacionalidad(nacionalidad_id)
   on delete cascade on update cascade
)engine=innoDB;
```

Bien, en primer lugar vamos a usar una nueva instrucción para nuestro código SQL, la palabra reservada **unique**, que nos permite que el valor de cada registro nuevo a insertar no sea repetido, osea que no haya otro registro con ese mismo nombre. Esto es muy útil en este campo, **email**, ya que no debería haber dos usuarios con el mismo email.

Por otro lado agregamos dos campos nuevos **fecha_alta** y **fecha_modificacion**, que nos permitirá guardar la fecha y hora en que se inserta un nuevo usuario y la fecha y hora cuando se modifica, respectivamente. Esto es muy útil para hacer seguimientos.

```
fecha_alta datetime not null,
fecha_modificacion datetime not null,
```

Por último creamos un atributo **nacionalidad_id**, que será una clave externa, que estará unida a la clave primaria nacionalidad_id, de la tabla **nacionalidad**. Vale aclarar que ambas deben coincidir en el tipo de dato y la cantidad de dígitos que soportan, además como la clave primaria **nacionalidad_id** de la tabla **nacionalidad** es **unsigned**, ésta también debe serlo.

```
nacionalidad_id tinyint(3) unsigned not null,
```

Ahora para finalizar la unión entre ambas tablas, debemos usar la palabra reservada **foreign key** y el nombre del atributo que será una clave externa, y **references** para indicar con qué tabla debemos unirla y entre paréntesis el nombre del atributo de esta tabla.

```
nacionalidad_id tinyint(3) unsigned not null,
foreign key(nacionalidad_id) references nacionalidad(nacionalidad_id)
```

Y con respecto a las líneas:

```
on delete cascade on update cascade
```

Esto nos permitirá que todo cambio y eliminación de registros en la tabla padre, en nuestro caso **nacionalidad**, afectará a los registros relacionados con la tabla hija, en nuestro caso **usuario**. Otro valor, en lugar de **cascade** podría ser **restrict**, el cual no permitirá modificar o eliminar registros de la tabla padre, si está relacionados con registro de la tabla hija.

Bien, seguimos creando la tabla **telefono**:

```
CREATE TABLE telefono(
   telefono_id int(10) unsigned not null auto_increment primary key,
```

```
   valor varchar(50) not null,
   usuario_id int(10) unsigned not null,
   foreign key (usuario_id) references usuario(usuario_id)
   on delete cascade on update cascade
)engine=innoDB;
```

Recordemos que la tabla **usuario** y **telefono** tienen una relación de 1 a 1, por esto la clave externa podría estar en cualquiera de las dos tablas, pero en este caso la agregaremos **telefono**, esto nos facilita un poco más las cosas, ya que no siempre un usuario tendrá un teléfono, pero cuando se registre uno nuevo, le indicamos como clave externa el id del usuario al que le pertenece.

Continuamos con la clave **interes**, para guardar los intereses que puede llegar a tener un usuario:

```
CREATE TABLE interes(
   interes_id tinyint(2) unsigned not null auto_increment primary key,
   valor varchar(100)
)engine=innoDB;
```

Pero cómo habíamos dicho que la tabla **interes** tendría una relación de n a n (muchos a muchos) con la tabla **usuario**, y que también este tipo de relaciones generan una nueva tabla intermedia. Entonces vamos a crear esa tabla:

```
CREATE TABLE usuario_interes(
   usuario_interes int(10) unsigned not null auto_increment primary key,
   usuario_id int(10) unsigned not null,
   interes_id tinyint(2) unsigned not null,
   foreign key(usuario_id) references usuario(usuario_id)
   on delete cascade on update cascade,
   foreign key(interes_id) references interes(interes_id)
   on delete cascade on update cascade
)engine=innoDB;
```

Como se ve en el ejemplo, la tabla **usuario_interes**, hará de puente, uniendo en cada registro un usuario y un interés, guardando el identificador para cada uno.

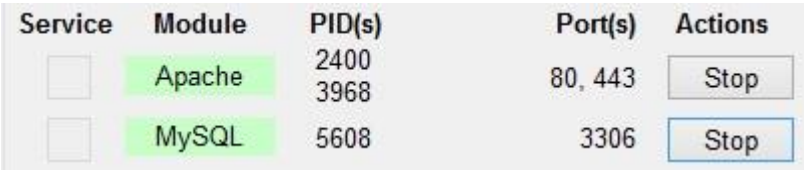
Bien, con esto tenemos nuestra primer base de datos, con sus tablas y las tres posibles relaciones. En la próxima publicación aprenderemos a hacer consultas en estas tablas.

Saludos!

Introducción a MySQL, registros

Como dijimos anteriormente, las tablas están compuestas por registros, por ejemplo en nuestra tabla **nacionalidad**, cada registro será, valga la redundancia, una nacionalidad. Pero para trabajar con esos registros debemos aprendernos las cuatro operaciones, que son insertar, modificar, eliminar y mostrar registros. Comencemos.

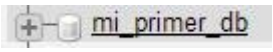
Vamos a asegurarnos como siempre que tanto nuestro servicio de Apache y MySQL estén encendidos. A esta altura ya deberíamos tenerlo bien claro, pero por las dudas:



Vamos a abrir el navegador y vamos a ir a:

```
https://localhost/phpmyadmin/
```

Y una vez dentro vamos a hacer clic en la base de datos con la que venimos trabajando hace tiempo que es **mi_primer_db**.



Insertar registros

Dentro del campo de ingreso SQL, (para ir, hay que pulsar el botón SQL) vamos a insertar nuestro primer registro.

La sintaxis para esto es la siguiente:

```
INSERT INTO tabla(atributo1, atributo2)
VALUES(valor_atributo1, valor_atributo2);
```

Sabiendo esto, vamos a insertar una nacionalidad con el siguiente código:

```
INSERT INTO nacionalidad(valor)
VALUES('Argentina');
```

Bien, en primer lugar, esta tabla tiene dos atributos, **nacionalidad_id** y **valor**, pero el primer atributo podemos ignorarlo, porque al ser **auto_increment**, como dijimos en la publicación anterior, se irá auto incrementados de a 1, 2, 3 y así sucesivamente, a medida que insertemos nuevos registros.

Otra cosa que hay que tener en cuenta es que el atributo **valor**, es un **varchar**, osea una cadena de texto, por tanto debe ir entre comillas:

```
'Argentina'
```

También podemos insertar de a varios registros:

```
INSERT INTO nacionalidad(valor)
VALUES('Brasil');
INSERT INTO nacionalidad(valor)
VALUES('Paraguay');
INSERT INTO nacionalidad(valor)
```

```
VALUES('Uruguay');
```

Otra forma de insertar registros es ingresando la sintaxis de **INSERT INTO (atributos)** e ir separando por coma los **values**:

```
INSERT INTO nacionalidad(valor)
VALUES
('Bolivia'),
('Chile'),
('Perú'),
('Colombia'),
('Ecuador'),
('Venezuela');
```

Ambas formas son correctas.

Sabiendo esto podemos insertar un registro que esté ligado a otro, por ejemplo podemos insertar un usuario con la nacionalidad 1, en mi caso 'Argentina':

```
INSERT INTO usuario(nombre, apellido, email, contrasena, fecha_alta,
fecha_modificacion, nacionalidad_id)
VALUES('Juan', 'Pérez', 'jperez@mail.com', '1234', sysdate(), sysdate(),
1);
```

En primer lugar, para guardar el valor de la **fecha_alta** y **fecha_modificacion**, usamos la función **sysdate()**, que nos devolverá la fecha y hora actual, en donde se está insertando el registro. También podríamos haber puesto la fecha y hora manualmente:

```
INSERT INTO usuario(nombre, apellido, email, contrasena, fecha_alta,
fecha_modificacion, nacionalidad_id)
VALUES('Juan', 'Pérez', 'jperez@mail.com', '1234', '2013-12-29 21:54:27',
'2013-12-29 21:54:27', 1);
```

Además para ingresar el valor de **nacionalidad_id**, ingresamos el id que le corresponde. En nuestro caso como es Argentina el id 1, éste será el valor.

Modificar registros

Así como insertamos registros, también podemos modificarlos mediante la sentencia **UPDATE**:

```
UPDATE tabla SET atributo1 = valor1, atributo2 = valor2;
```

Por ejemplo si quisiéramos modificar el registro que acabamos de insertar:

```
UPDATE usuario SET nombre = 'Pedro', apellido = 'Rodríguez',
fecha_modificacion = sysdate();
```

El problema es que este código SQL va a modificar todos los registros de la tabla usuario, con el nombre 'Pedro', apellido 'Rodríguez' y también actualizará la fecha de modificación. Para modificar un registro específico, debemos agregar una condición con la

palabra reservada **WHERE** y la condición. Por ejemplo si quisiéramos modificar el registro con el **usuario_id** que sea 1:

```
UPDATE usuario SET nombre = 'Pedro', apellido = 'Rodríguez', fecha_modificacion = sysdate() WHERE usuario_id = 1;
```

En la mayoría de los casos se utiliza una condición para **UPDATE**, de esta manera podemos modificar un registro específico o varios registros que cumplan una condición.

Por ejemplo con esto podemos modificar todos los usuarios que se tengan como nombre 'Pedro':

```
UPDATE usuario SET nombre = 'Juan', apellido = 'Rodríguez', fecha_modificacion = sysdate() WHERE nombre = 'Pedro';
```

Eliminar registros

Si lo que queremos es eliminar registros debemos usar la palabra reservada **DELETE**:

```
DELETE FROM tabla;
```

Obviamente al igual que **UPDATE** podemos especificar una condición para eliminar uno o más registros. Por ejemplo, si quisiéramos eliminar el que acabamos de crear y modificar:

```
DELETE FROM usuario WHERE usuario_id = 1;
```

Mostrar registros

Bien, ya sabemos cómo insertar, modificar y eliminar registros, pero esto es sólo una parte del trabajo con registros. También nosotros debemos tener la posibilidad de recuperar esos registros, buscar uno o varios que cumplan una condición, ordenarlos, etc.

Por ejemplo supongamos que queremos recuperar todos los registros de la tabla **nacionalidad**:

```
SELECT * FROM nacionalidad;
```

El caracter ***** nos devolverá todas las columnas de la tabla, en este caso son sólo dos, pero en consultas con muchas columnas y muchos registros, es aconsejable sólo llamar a las columnas con los valores que necesitamos recuperar, para así hacer la consulta más liviana. Por ejemplo para recuperar sólo la columna **valor**:

```
SELECT valor FROM nacionalidad;
```

Esto nos devolverá:

Argentina
Brasil
Paraguay
Uruguay
Bolivia
Chile
Perú
Colombia
Ecuador
Venezuela

También podemos agregar condiciones, por ejemplo recuperar el registro con el id 2, que en mi caso es Brasil:

```
SELECT valor FROM nacionalidad WHERE nacionalidad_id = 2;
```

O también:

```
SELECT valor FROM nacionalidad WHERE valor = 'Brasil';
```

Tener en cuenta que al ser una cadena debemos usar comillas como dijimos antes.

También podemos ordenar nuestros registros, por ejemplo por el nombre de la nacionalidad, mediante la palabra reservada **order by**:

```
 SELECT nacionalidad_id, valor FROM nacionalidad ORDER BY valor;
```

O en forma descendente, con la palabra **DESC** detrás del nombre del atributo, osea en lugar de que el primero que se muestre sea ‘Argentina’, sea ‘Venezuela’, que en orden alfabéticamente, son el primero y último registro respectivamente:

```
SELECT nacionalidad_id, valor FROM nacionalidad ORDER BY valor DESC;
```

Otra posibilidad es la de limitar los registros, por ejemplo que me traiga los 5 primeros que encuentre:

```
SELECT nacionalidad_id, valor FROM nacionalidad LIMIT 5;
```

O seleccionar registros buscándolos por coincidencia, por ejemplo si quisiéramos recuperar las nacionalidades que contengan la cadena ‘guay’, que son ‘Paraguay’ y ‘Uruguay’. Aquí tendremos que usar la palabra **LIKE**, en lugar del igual (=)

```
SELECT nacionalidad_id, valor FROM nacionalidad WHERE valor LIKE
'%guay%';
```

Con respecto al caracter **%**, en SQL ese caracter es un comodín. Lo que hará la consulta en realidad es buscar todas las nacionalidades en donde su valor esté formado por en juego de caracteres ‘guay’, y gracias al comodín aclaramos que esa cadena sea buscada en el medio, osea sin importar los caracteres que se encuentren atrás o delante de la misma.

También podríamos buscar todas las nacionalidades, que por ejemplo empiecen con ‘B’, osea ‘Bolivia’ y ‘Brasil’:

```
SELECT nacionalidad_id, valor FROM nacionalidad WHERE valor LIKE 'B%';
```

En este caso ingresamos el caracter comodín sólo delante, para especificar que el primer caracter debe ser ‘B’, y el resto puede ser cualquier cosa

O buscar todas las nacionalidades que terminen con ‘a’, que en nuestro caso serán: ‘Argentina’, ‘Bolivia’, ‘Colombia’ y ‘Venezuela’:

```
SELECT nacionalidad_id, valor FROM nacionalidad WHERE valor LIKE '%a';
```

Incluso podemos hacer una consulta bastante trabajada con todo lo que acabamos de aprender de **SELECT**, por ejemplo recuperar los tres primeros registros de todas las nacionalidades que tengan la letra ‘e’ y ordenarlos en forma descendente:

```
SELECT valor FROM nacionalidad WHERE valor LIKE '%e%' ORDER BY valor DESC
LIMIT 3;
```

El resultado será:

Venezuela
Perú
Ecuador

Unión de tablas

Supongamos que queremos hacer un **SELECT**, pero en lugar de traer registros de una tabla, queremos combinar consultas con dos o más registros de tablas diferentes. Por ejemplo, queremos recuperar los usuarios de nuestra base de datos, pero en lugar de traer el id de la nacionalidad, queremos ver el texto, el nombre de la nacionalidad. Para conseguir eso debemos combinar el registro de cada usuario con el registro de la nacionalidad, con la que el usuario está conectado.

En primer lugar vamos a insertar un listado de usuarios con distintas nacionalidades:

```
INSERT INTO usuario(nombre, apellido, email, contrasena, fecha_alta,
fecha_modificacion, nacionalidad_id)
VALUES
('Ricardo', 'Caruso Lombardi', 'caruso@mail.com', '1234', sysdate(),
sysdate(), 1),
('Susana', 'Giménez', 'susana@mail.com', '1234', sysdate(), sysdate(),
1),
('Wilson', 'Sanguinetti', 'wilson@mail.com', '1234', sysdate(),
sysdate(), 4),
('Ronaldo', 'Santos', 'ronaldo@mail.com ', '1234', sysdate(), sysdate(),
2),
('Marisa', 'Santa Cruz', 'marisa@mail.com', '1234', sysdate(), sysdate(),
3);
```

Como se ve en el ejemplo, insertamos cinco usuarios, de los cuales dos tienen nacionalidad Argentina, y otros tres con nacionalidad de Uruguay, Paraguay y Brasil.

Entonces para recuperar los usuarios por su nombre y apellido, y el atributo **valor** de la tabla **nacionalidad**, con la que le corresponde al usuario, deberíamos hacer esto:

```
SELECT usuario.nombre, usuario.apellido, nacionalidad.valor FROM usuario
LEFT JOIN nacionalidad ON usuario.nacionalidad_id =
nacionalidad.nacionalidad_id;
```

En primer lugar, hacemos un **SELECT** de una tabla como venimos haciendo, luego agregamos la unión mediante **LEFT JOIN** y el nombre de la tabla con la que queremos unir la consulta, en este caso **nacionalidad**.

La razón por la cual usamos el nombre de la tabla más un punto y el nombre del atributo, se debe a que cuando unimos tablas, éstas pueden tener atributos con el mismo nombre, por esto es una buena práctica al unir tablas, llamar a los atributos con el nombre de la tabla y el punto adelante.

Por último, la palabra reservada **ON**, nos permitirá indicar cuáles son los atributos que deben coincidir, en nuestro caso los atributos con el mismo nombre **nacionalidad_id**, de ambas tablas.

Esta consulta nos devolverá:

Ricardo	Caruso Lombardi	Argentina
Susana	Giménez	Argentina
Wilson	Sanguinetti	Uruguay
Ronaldo	Santos	Brasil
Marisa	Santa Cruz	Paraguay

Ahora, supongamos que quisiéramos realizar esto pero a la inversa,osea que en lugar de recuperar los usuarios y sus nacionalidades, queremos recuperar las nacionalidades y los usuarios que pertenecen a las mismas. Para esto tenemos dos posibilidades, una es rotar en la consulta la tabla **usuario** y **nacionalidad**, de modo que la que quede del lado izquierdo (**LEFT JOIN**) sea **nacionalidad**. O bien, modificar **LEFT JOIN** por **RIGHT JOIN**:

```
SELECT usuario.nombre, usuario.apellido, nacionalidad.valor FROM usuario
RIGHT JOIN nacionalidad ON usuario.nacionalidad_id =
nacionalidad.nacionalidad_id;
```

Esta consulta nos devolverá lo siguiente:

Ricardo	Caruso Lombardi	Argentina
Susana	Giménez	Argentina
Ronaldo	Santos	Brasil
Marisa	Santa Cruz	Paraguay

Wilson	Sanguinetti	Uruguay
NULL	NULL	Bolivia
NULL	NULL	Chile
NULL	NULL	Perú
NULL	NULL	Colombia
NULL	NULL	Ecuador
NULL	NULL	Venezuela

Si observamos bien, la consulta nos devolverá todas las nacionalidades, en el caso de Argentina veremos dos registros, ya que hay dos usuarios con esta nacionalidad, en el caso de Brasil, Paraguay y Uruguay nos devolverá tan sólo un registro, ya que cada uno de estos tiene un sólo usuario con esa nacionalidad. Pero, también nos devolverá aquellas nacionalidades, que aún no tienen usuarios asociados a las mismas, una vez por cada una de estas y con el valor de nombre y apellido en **NULL**, claro está, como acabo de decir, no tienen usuarios asociados.

Si bien en la tabla **usuario**, la nacionalidad del mismo es obligatoria, podría existir la posibilidad que no lo sea, y que algunos usuarios hayan sido insertados sin nacionalidad, y en ese caso, al tirar la primer consulta (**LEFT JOIN**) hubiese pasado lo mismo que pasó con la última (**RIGHT JOIN**), nos hubiese devuelto usuarios con **nacionalidad_id** con el valor **NULL**, en aquellos registros que hagan referencia a usuarios sin nacionalidad.

Si lo que en cambio queremos es realizar una consulta que no depende de una tabla u otra, sino que nos devuelva todos aquellos registros en que coincidan, entonces deberíamos usar **INNER JOIN**. Entonces podríamos modificar la consulta finalmente por:

```
SELECT usuario.nombre, usuario.apellido, nacionalidad.valor FROM usuario INNER JOIN nacionalidad ON usuario.nacionalidad_id = nacionalidad.nacionalidad_id;
```

En este caso no nos devolverá aquellos usuarios que no tengan nacionalidad (si existiera esa posibilidad), o las nacionalidades que no estén ligadas a ningún usuario. Los registros encontrados serán sólo usuarios con nacionalidad y nacionalidades que pertenezcan a usuarios.

Bueno, con esto terminamos nuestra introducción a MySQL. Y en el capítulo que viene vamos retomar PHP con MySQL.

Recordá que si querés profundizar en ciertos temas, siempre podés consultar a la página oficial de MySQL:

https://www.mysql.com/

Saludos!

Conectar PHP con MySQL

Ahora que ya sabemos programar en PHP y realizar consultas con MySQL, podemos unir ambas tecnologías para conectar a PHP con MySQL y realizar consultas desde nuestra aplicación. Para ello PHP nos provee de una gran variedad de funciones que nos permitirán realizar esto. Así que empecemos.

Para conectar nuestra aplicación a MySQL necesitamos cuatro datos básicamente. En primer lugar el host o IP del servidor al cual conectaremos, en nuestro caso tanto Apache como MySQL se encuentran en el mismo servidor, nuestra máquina, así que será 'localhost' o su IP, '127.0.0.1'. El nombre del usuario de base de datos, éste será 'root' y la contraseña, que por defecto será una cadena vacía. Y finalmente el nombre de la base de datos a la cual vamos a conectarlos, que será lo que venimos trabajando hasta ahora, 'mi_primer_db'

Todo esto suponiendo que vamos a conectarnos a nuestro localhost, claro ésta. Si vas a conectar PHP con MySQL a un servidor remoto deberías solicitar estos cuatro datos que acabamos de mencionar: el host, usuario, contraseña y nombre de la base de datos, a menos que seas vos el que administre el servicio de MySQL y deberías tenerlos.

Conectar PHP con MySQL

Existen varias formas de conectar PHP con MySQL, una de ellas es mediante PDO, especificado en una publicación que hice hace aproximadamente un año: Php orientado a objetos, parte 10: PDO, Conectarse a una base de datos. Pero ésta está orientada a programadores con conocimientos medios o avanzados de PHP, y también orientados a objetos. En esta ocasión veremos cómo conectarnos mediante la función: **mysqli_connect()**.

Esta función recibirá seis parámetros, aunque los dos últimos no son obligatorios. El primero será el host, el segundo el usuario, el tercero la contraseña y el cuarto el nombre de la base de datos. Los dos últimos vamos a omitirlos, ya que no son necesarios, estos son el puerto y el socket, quinto y sexto respectivamente.

Vamos a conectarnos entonces:

```
<?php
$conexion = mysqli_connect('localhost', 'root', '', 'mi_primer_db');
mysqli_close($conexion);
?>
```

Como se ve en el ejemplo, la primer función: **mysqli_connect()** conecta nuestra aplicación a MySQL, con los valores que mencionamos antes y guardamos dicha conexión en una variable, para así mantenerla a lo largo de nuestro script. Finalmente con la función **mysqli_close()** realizamos el proceso inverso que es desconectar nuestra aplicación, esta función recibirá un parámetro que será justamente la conexión que

queremos cerrar. Esto último es muy importante a tener en cuenta, ya que lo recomendable es conectarnos a nuestra base de datos, realizar las consultas que debemos hacer, como recuperar registros y guardarlos en una variable, y finalmente cerrar la conexión para liberar recursos.

Posible error de conexión

Hablamos en algún momento que existen varios tipos de errores y motivos de los mismos en PHP, pero también señalamos que estos errores pueden darse por un error del programador, o por un error ajeno al programador, pero igual éste debe estar preparado para tapar estos baches.

Cuando conectamos PHP con MySQL, al tratarse de dos servicios que si bien, conviven entre sí, son servicios separados, y uno de estos puede no estar funcionando por diversos motivos. En el caso de que PHP no pueda conectarse a MySQL nosotros deberíamos interrumpir el script para sí evitar que el mismo continúe y genere nuevos errores, provocados por este primero. Para ello vamos a modificar nuestro script de la siguiente forma:

```
<?php
$conexion = mysqli_connect('localhost', 'root', '', 'mi_primer_db') or
die('Error al intentar conectarse a la base de datos.');
echo 'Conectado correctemente';
mysqli_close($conexion);
?>
```

En primer lugar conectamos nuestra aplicación a MySQL, pero antes de finalizar la línea donde realizamos dicho proceso, agregamos el bloque de código:

```
or die('Error al intentar conectarse a la base de datos.');
```

El cual se ejecutará si al intentar conectarse con la base de datos algo falla. La función **die()**, al igual que **exit()**, finalizará el script, por tanto todo lo que siga será omitido por nuestro servidor. Podemos simular este error apagando el servicio de MySQL o ingresando algún valor de la función en forma errónea.

Para ejemplificar esto, podemos imaginarnos esto, debemos conectar nuestra aplicación a MySQL y tirar una consulta para recuperar una serie de registros, pero si la conexión falla, obviamente no podremos recuperar esos registros, entonces PHP nos devolverá dos errores, al intentar conectar con la base de datos, y al realizar una consulta a una base de datos con la que no hemos podido establecer una comunicación. Como vimos, de esta forma, al encontrar el primer error el script finalizará y mostramos un error por pantalla antes de terminar.

Consultas

Para realizar una consulta, debemos usar la función **mysqli_query()**, que recibirá dos parámetros, la conexión y la consulta misma.

Por ejemplo podemos realizar un **INSERT**:

```
<?php
$conexion = mysqli_connect('localhost', 'root', '', 'mi_primer_db') or
die('Error al intentar conectarse a la base de datos.');
mysqli_query($conexion, 'SET NAMES "utf8"');
$valor = 'Música';
$consulta = "INSERT INTO interes(valor) VALUES('$valor')";
if(mysqli_query($conexion, $consulta)){
   echo 'Registro insertado correctamente.';
}else{
   echo 'Error: ' . mysqli_error($conexion);
}
mysqli_close($conexion);
?>
```

Vamos a analizar el código:

En primer lugar conectamos con la base de datos, ya hemos visto esto:

```
$conexion = mysqli_connect('localhost', 'root', '', 'mi_primer_db') or
die('Error al intentar conectarse a la base de datos.');
```

En segundo lugar disparamos una consulta mediante **mysqli_query()** con el código SQL: ‘SET NAMES “utf8″‘, esto es necesario cuando debemos enviar datos que pueden contener caracteres especiales. Lo que haremos a continuación es insertar el nombre de un nuevo interés, el cual puede contener algún tilde o ñ, por tanto debemos especificar previamente a MySQL que será una conexión utf8, que es la que soportará este tipo de caracteres.

Luego creamos dos variables, una que guardará el nombre del interés y otro la consulta:

```
$valor = 'Música';
$consulta = "INSERT INTO interes(valor) VALUES('$valor')";
```

Y finalmente mediante la función **mysqli_query()** ejecutamos la consulta, la cuál nos devolverá **true** si la misma se ha podido realizar o **false** si no ha sido así:

```
if(mysqli_query($conexion, $consulta)){
   echo 'Registro insertado correctamente.';
}else{
   echo 'Error: ' . mysqli_error($conexion);
}
```

Notar que si la consulta no ha podido realizarse, mostramos que ha habido un error, concatenándolo con el resultado que nos devuelve la función: **mysqli_error()**, esta función será la que nos permita recuperar los errores ocasionados por MySQL.

Y finalmente cerramos la conexión:

```
mysqli_close($conexion);
```

Ahora supongamos que quisiéramos recuperar el id del registro que acabamos de insertar, deberíamos hacer esto:

```
<?php
$conexion = mysqli_connect('localhost', 'root', '', 'mi_primer_db') or
die('Error al intentar conectarse a la base de datos.');
mysqli_query($conexion, 'SET NAMES "utf8"');
$valor = 'Cine';
$consulta = "INSERT INTO interes(valor) VALUES('$valor')";
if (mysqli_query($conexion, $consulta)) {
   echo 'Registro insertado correctamente con el id: ' .
mysqli_insert_id($conexion);
} else {
   echo 'Error: ' . mysqli_error($conexion);
}
mysqli_close($conexion);
?>
```

La función **mysqli_insert_id()** nos devolverá el id o clave primaria del último registro que acabamos de insertar.

El mismo proceso se utiliza para realizar consultas **UPDATE** y **DELETE**. Sin embargo, imaginemos que queremos hacer un **SELECT**, por ejemplo para recuperar los países de nuestra base de nuestra tabla **nacionalidad**:

```
<?php
$conexion = mysqli_connect('localhost', 'root', '', 'mi_primer_db') or
die('Error al intentar conectarse a la base de datos.');
$consulta = "SELECT nacionalidad_id, valor FROM nacionalidad ORDER BY
valor";
$resultado = mysqli_query($conexion, $consulta);
while($registro = mysqli_fetch_array($resultado)){
   echo $registro['valor'] . '<br />';
}
mysqli_close($conexion);
?>
```

Gracias a la función **mysqli_fetch_array()** podremos recuperar los registros que nos devuelva la consulta en forma de array asociativo, sin embargo cada vez que llamemos a esta función la misma nos devolverá de a un registro, por eso al incluirlo dentro de un **while**, nosotros repetiremos esta acción hasta que ya no queden registros y el ciclo finalice.

Sql injection

Hasta ahora hemos hecho consultas mediante PHP, pero debemos tener en cuenta una cosa. Las consultas no siempre serán escritas 100% por nosotros, osea, con esto me refiero a que, muchas veces los valores que se ingresen en éstas, vendrán de formularios que deberán llenar los visitantes de nuestro sitio. Por ejemplo, un formulario para registrar un usuario,

los datos que ingrese el usuario como nombre y apellido, serán enviados al servidor e incluidos dentro de la consulta.

Primer problema: el usuario puede ingresar valores que rompan nuestra consulta. Por ejemplo, si la consulta es:

```
SELECT valor FROM nacionalidad WHERE valor = '$nacionalidad';
```

Suponemos que el usuario ingresó en un campo de formulario: 'Argentina', y se recupera ese valor desde una variable POST, entonces la consulta que se envíe a MySQL será:

```
SELECT valor FROM nacionalidad WHERE valor = 'Argentina';
```

Hasta acá todo muy lindo. Pero, qué pasa si el usuario ingresa un caracter que rompa ese **SELECT**, por ejemplo envíe el valor: "Argentina', osea una comilla simple y nombre. Entonces la consulta quedará:

```
SELECT valor FROM nacionalidad WHERE valor = ''Argentina';
```

Esto se romperá.

Segunda problema: Al permitirle al usuario ingresar datos en la consulta, como acabamos de ver, esto da a lugar a que visitantes malintencionados, ingresen texto que pueda llegar a alterar el verdadero propósito de nuestra consulta. Eso se llama **sql injection**. No voy dar ejemplos de **sql injection** debido a que pretendo enseñar a evitar ataques, no a provocarlos. Pero este método ha ocasionado muchos dolores de cabeza de los hackers a los programadores, incluso mediante el **sql injection**, se han alterado consultas SQL que provocaron que visitantes puedan acceder a información a la cual no estaban autorizados.

Teniendo en cuenta estos dos puntos, debemos tener en cuenta que esos valores que el visitante ingresa, deben ser procesados y formateados antes de realizar una consulta con los mismos.

Supongamos que el usuario se registra:

```
<?php
$conexion = mysqli_connect('localhost', 'root', '', 'mi_primer_db') or
die('Error al intentar conectarse a la base de datos.');
mysqli_query($conexion, 'SET NAMES "utf8"');
//Datos ingresados por el usuario.
$nombre = 'Fernando';
$apellido = 'Gaitán';
$email = 'fgaitan@mail.com';
$contrasena = md5('1234');
$nacionalidad_id = 1;
//Escapar valores ingresados.
$nombre = mysqli_real_escape_string($conexion, $nombre);
$apellido = mysqli_real_escape_string($conexion, $apellido);
$email = mysqli_real_escape_string($conexion, $email);
$contrasena = mysqli_real_escape_string($conexion, $contrasena);
$nacionalidad_id = mysqli_real_escape_string($conexion,
$nacionalidad_id);
```

```
$consulta = "INSERT INTO usuario(nombre, apellido, email, contrasena,
fecha_alta,   fecha_modificacion,   nacionalidad_id)   VALUES('$nombre',
'$apellido',    '$email',    '$contrasena',    sysdate(),    sysdate(),
$nacionalidad_id)";
if(mysqli_query($conexion, $consulta)){
   echo 'El usuario ha sido insertado correctamente.';
}else{
   echo 'Error: ' . mysqli_error($conexion);
}
mysqli_close($conexion);
?>
```

Vamos a analizar el código:

```
$nombre = 'Fernando';
$apellido = 'Gaitán';
$email = 'fgaitan@mail.com';
$contrasena = md5('1234');
$nacionalidad_id = 1;
```

Aquí simularemos una serie de datos ingresados por el usuario para registrarse. Notar que la contraseña pasará por un proceso de encriptación gracias a la función **md5()**, así podremos proteger las contraseñas en nuestra base de datos, convertidas en cadena disfrazadas.

Y ahora lo que nos interesa:

```
$nombre = mysqli_real_escape_string($conexion, $nombre);
$apellido = mysqli_real_escape_string($conexion, $apellido);
$email = mysqli_real_escape_string($conexion, $email);
$contrasena = mysqli_real_escape_string($conexion, $contrasena);
$nacionalidad_id          =          mysqli_real_escape_string($conexion,
$nacionalidad_id);
```

La función **mysqli_real_escape_string()**, le dará un formato a nuestros valores ingresados para así corregir aquellas combinaciones de caracteres que puedan romper la consulta o bien alterarla.

Si ejecutamos el script se insertará un usuario.

Ahora supongamos que queremos verificar si los datos que ingresó el mismo son correctos para así loguearlo:

```
<?php
$conexion = mysqli_connect('localhost', 'root', '', 'mi_primer_db') or
die('Error al intentar conectarse a la base de datos.');
mysqli_query($conexion, 'SET NAMES "utf8"');
//Datos ingresados por el usuario.
$email = 'fgaitan@mail.com';
$contrasena = md5('1234');
//Escapar valores ingresados.
$email = mysqli_real_escape_string($conexion, $email);
$contrasena = mysqli_real_escape_string($conexion, $contrasena);
```

```
$consulta = "SELECT usuario_id FROM usuario WHERE email = '$email' AND
contrasena = '$contrasena'";
$resultado = mysqli_query($conexion, $consulta);
$usuario = mysqli_fetch_array($resultado);
if($usuario){
   echo 'Se ha logueado un usuario con el id: ' . $usuario['usuario_id'];
}else{
   echo 'Datos incorrectos.';
}
mysqli_close($conexion);
?>
```

Aquí repetimos el mismo proceso, formateamos los datos ingresados por el usuario, para así evitar **sql injection**, y verificamos con un **if** si la consulta nos ha devuelto algún registro:

```
if($usuario){
   echo 'Se ha logueado un usuario con el id: ' . $usuario['usuario_id'];
}else{
   echo 'Datos incorrectos.';
}
```

Saludos!
