APUNTES PHP

José Juan Urrutia Milán

Reseñas

- Curso PHP: https://www.youtube.com/playlist?list=PLU8oAlHdN5BkinrODGXToK9oPAlnJxmW_
- Instalación del software (min 5:55 y vídeo 3 del mismo curso): https://www.youtube.com/watch?v=tXxOAXP-gkg&list=PLU8oAlHdN5BkinrODGXToK9oPAlnJxmW_&index=2
- Uso PHPmyadmin: https://www.youtube.com/watch?v=LKrBvE3yAZo&list=PLU8oAlHdN5BkinrODGXToK9oPAlnJxmW_&index=33

Siglas/Vocabulario

- **IDE:** Entorno de Desarrollo Integrado.
- **BBDD:** Bases de Datos.
- **SGBD:** Sistemas Gestores de Bases de Datos.
- **GUI:** Graphic User’s Interface. Interfaz gráfica de usuario.

Leyenda

Cualquier abreviatura o referencia será subrayada.
Cualquier ejemplo será escrito en **negrita.**
Cualquier palabra de la que se pueda prescindir irá escrita en cursiva.
Cualquier abreviatura viene explicada a continuación:

Abreviaturas/Referencias:

- 123: Hace referencia a cualquier número.
- nombre: Hace referencia a cualquier palabra/cadena de caracteres.
- a: Hace referencia a cualquier caracter.
- cosa: Hace referencia a cualquier número/palabra/cadena.
- Tipo_var o **…** : Hace referencia a cualquier tipo de variable primitiva o de tipo String.
- nombre_var: Hace referencia a cualquier nombre que se le puede dar a una variable.
- código: Hace referencia a cualquier instrucción. (Se usará para indicar dónde se podrá inscribir código.)
- variable: Hace referencia a cualquier variable.
- condición: Hace referencia a cualquier condición. Entiéndase por condición, una afirmación que devuelve un true o un false. Ej: (variable == 123). *Una variable del tipo boolean puede ser usada como una condición.

Índice

strcasecmp($string1, $string2)

Título II: Operadores

Concatenación

Lógicos

Aritméticos

Comparación

Título III: Funciones predefinidas

rand()

rand($min, $max)

pow($base, $exp)

round($n)

round($n, $precision)

strtolower($string)

strtoupper($string)

ucwords($string)

header($pagina)

Refrescar página actual

date($formato)

empty($var)

isset($var)

Título IV: Castings

Implícitos

Explícitos

Título V: Condicionales

if

Operador ternario

switch

Título VI: Bucles

Capítulo IV: Cookies

Qué es una Cookie

Título I: Creación de cookies

Tiempo de vida

Directorios de lectura

Título II: Lectura de cookies

Comprobación de cookies

Título III: Eliminación de cookies

Capítulo V: Imágenes

Introducción

Curso destinado al aprendizaje del lenguaje PHP, así como conexiones con BBDD MySQL.
PHP es un lenguaje del lado del servidor, por lo que necesitaremos usar un servidor web para ejecutar código PHP o un paquete que lo simule, como el paquete WAMP.

Los archivos PHP se guardan con extensión .php .
Los archivos PHP (si trabajamos con WAMP), los tenemos que guardar en: C:/wamp64/www/Nueva Carpeta
*Podemos cambiarle el nombre a esta Nueva Carpeta.
No podemos incluir código PHP dentro de un archivo .html
Dentro de un archivo .php podemos incluir código HTML, siempre que deseemos incluir una interfaz gráfica, aunque no siempre es necesario. También podemos incluir código CSS.
Para ver un archivo PHP, debemos dirigirnos al navegador y escribir la ruta: http://localhost/Nueva%20Carpeta/nombrearchivo.php
Si nuestro archivo se llama index, no hace falta indicar nombre de archivo.

PHP es un lenguaje orientado a objetos.
PHP es case sensitive.
PHP no sigue la nomenclatura camelCase.
Cada sentencia PHP termina con un punto y coma: “;” .
Los textos en PHP se indican entre comillas dobles.
No es lo mismo una comilla doble que una simple.

Capítulo I: Conceptos básicos

Comentarios

Texto que el compilador ignora.
Para indicar un comentario de una línea, especificamos dos barras: **//Hola**.
Para indicar un comentario de varias líneas, especificamos /* y */ al final:
/*Hola
como
estas*/
Idéntico a Java.

Colocación del código

El código PHP se introduce en un archivo .php con código HTML dentro de la etiqueta **<body>.**
El código PHP debe ir dentro de:
<?php
código
?>

Podemos encontrar diferentes etiquetas PHP en un mismo documento.

*En caso de que nuestro documento no tenga código HTML, tenemos que seguir especificando nuestro código dentro de una etiqueta:
<?php código; ?> .
También podemos incluir código PHP dentro del **head .
***Podemos incluir código PHP dentro de componentes HTML.

print

Imprime en pantalla el texto indicado a continuación:
print "Hola mundo";
(ver Apuntes HTML, Capítulo II, Título II, tildes).

*Se pueden especificar en él etiquetas HTML:
**print “Hola
mundo”;**

print es una función y devuelve siempre el valor 1.

echo

Imprime en pantalla el texto indicado a continuación:
echo “Hola mundo”;
(ver Apuntes HTML, Capítulo II, Título II, tildes).
*Se pueden especificar en él etiquetas HTML:
**echo “Hola
mundo”;**

echo es una expresión y, a diferencia de **print** , admite imprimir diferentes variables a la vez:
echo $nombre,$edad; Salida: Juan18

echo se suele utilizar más que **print** , además de que es más eficiente.

*Si se utiliza **echo** en el mismo fichero que el código HTML, esta salida se añadirá al final del último elemento HTML.
Por el contrario, si el **echo** se encuentra en un fichero .php independiente al código HTML, la página web entera se reemplazará por una vacía con este **echo**.
Para que esto no suceda, usar **include()** dentro del fichero haciendo referencia al fichero externo y así incluyendo dentro de la página web el **echo**.

include($archivo)

Nos permite incluir en nuestro código PHP otro código PHP externo que hayas desarrollado tú u otros programadores:
include(“archivo.php”);
Con esto, podemos usar en nuestro código funciones del archivo **archivo.php** y acceder a todas sus variables.

*Si no puede incluir el archivo, continúa con la ejecución del código PHP.
Podemos incluir una etiqueta **<?php?> dentro del **body** HTML con una única instrucción **include()** hacia un archivo HTML. De esta forma, estaremos creando una especie de "iframe" con PHP.
***Para ir a la carpeta raíz del programa, indicar .. antes de la ruta: **"../carpeta/archivo.php"**
****Al indicar **include** , es como si el archivo que se incluye está en la misma ruta que el archivo que ejecuta el **include()** .

include_once($archivo)

Idéntico a **include** salvo que si ya se ha incluido el archivo con **include**, no se ejecuta.

require($archivo)

Realiza la misma función que **include** , pero en caso de que no se pueda incluir el archivo, sale del código PHP.
*Podemos usar **require** con la misma sintaxis que **echo**:
require "archivo.php";
Para ir a la carpeta raíz del programa, indicar .. antes de la ruta: **"../carpeta/archivo.php"

require_once($archivo)

Idéntico a **require** salvo que si ya se ha incluido el archivo con **require**, no se ejecuta.

exit()

Sale del código PHP.

die($txt)

Imprime el mensaje **$txt** y sale del código PHP.

Procesar formularios con PHP

Podemos procesar la información de los formularios con PHP incluyendo en la etiqueta **<form>** el atributo **action** , que igualaremos al nombre de nuestro documento .php y el atributo **method** que igualaremos a **get** o a **post**:
<form action="archivo.php" method="get"> o:
<form action="archivo.php" method="post">
*Si el script php está en el mismo archivo que el código HTML, no es necesario especificar ningún **action** o indicar **"<?php echo $_SERVER["PHP_SELF"] ?>"** para refrescar la página actual al pulsar el botón.

Dentro de este documento, podemos usar el método **isset()** para obtener la información de los campos de texto:

```
if(isset($_POST["enviar"])){     //Si el name=enviar ha sido pulsado
        $nombre = $_POST["nombre"];       /*Captura el value del campo de texto con name=nombre.*/
        $edad = $_POST["edad"];        /*Captura el value del campo de texto con name=edad.*/
        código;
}
```

*utilizar **$_GET** si **method="get"** .
*Podemos prescindir del **if** y de su condición.

Título I: Variables

En PHP, todos los nombres de variables tienen que comenzar por el símbolo del dólar: "$" .
En PHP no es necesario especificar el tipo de dato, sino que se asignan dinámicamente. Es similar a JavaScript o python en ese sentido.

Declaración

Para declarar una variable, simplemente especificamos su nombre:
$nombre;
*Aunque esto no es habitual en PHP.

Inicialización

Para inicializar una variable o darle un valor, indicamos su nombre y un valor:
$nombre = valor ;
$nombre = "Juan";
$edad = 18;

Tipos de valores

En PHP, existen 3 tipos de valores:
- Strings: o cadenas de texto.
- Numéricos: Números enteros y decimales.
- Booleanos: **true** o **false** .

*En PHP, **true = 1** y **false = 0** .

Concatenación

Para concatenar diferentes elementos, usamos el punto:
print "Mi nombre es " . $nombre . " y tengo " . $edad . " años";
Salida: Mi nombres es Juan y tengo 18 años

PHP permite incluir el nombre de las variables dentro de Strings:
print "Mi nombres es $nombre y tengo $edad años";
Salida: Mi nombres es Juan y tengo 18 años

Si especificamos comillas simples en vez de dobles, se imprimirá la información de forma estricta:
print 'Mi nombres es $nombre y tengo $edad años';
Salida: Mi nombres es $nombre y tengo $edad años

Constantes / define($nombre, $valor)

Para declarar una constante, debemos usar la función **define()**.
- Convención: Por convención, el nombre de las constantes debe ir en mayúsculas.
- El nombre de las constantes no lleva el símbolo del dólar.
- El ámbito de las constantes es global por defecto.
- Las constantes no se pueden redefinir.
- Las constantes sólo pueden almacenar escalares.

define("EDAD", 15);
echo EDAD;

*Podemos añadir a la función **define** el valor **true** como tercer parámetro para declarar que la variable se podrá usar tanto en minúsculas como en mayúsculas, aunque siempre se recomienda la segunda:

define("EDAD", 15, true);
echo EDAD;
echo edad;

Ámbito

local

Declarada dentro de una función. Accesible y visible desde dentro de esta función.
No podemos acceder a variables locales dentro de funciones. Por lo que para acceder a estas las tenemos que declarar globales dentro de la función:

$nombre = "Juan";
function escribir(){global $nombre; echo $nombre;}

*Otra alternativa sería pasar la variable por un parámetro.
Para declarar una variable local, la declaramos de forma normal.

global

Declarada en cualquier lugar del código. Accesible y visible desde cualquier lugar del código.
Para declarar una variable global, especificar **global** antes de declararla.

super global

Declarada como array. Accesible y visible desde fuera del script PHP (se puede acceder a ella desde otros scripts).

Variables estáticas

Las variables estáticas son variables que no resetean su valor en cada llamada de una función.
Para declarar una variable estática, indicamos la palabra **static** antes de la declaración de esta variable:
static $edad;

Escape de carácter

Si dentro de un string indicamos una barra invertida "\" , estamos indicando al string que el siguiente carácter no pertenece al string. Esto nos puede resultar útil a la hora de crear comillas anidadas y no querer usar las comillas simples:
echo " Hola ";
echo "<p class=\"parrafo\">Hola ";

Comparación de Strings

strcmp($string1, $string2)

Devuelve un 0 si los valores de strings coinciden y un 1 si no coinciden. Diferencia entre minúsculas y mayúsculas.
$resultado = strcmp("hola", "HOLA"); //resultado = 1
*Como va al revés de la lógica, se recomienda invertirla.

strcasecmp($string1, $string2)

Devuelve un 0 si los valores de strings coinciden y un 1 si no coinciden. No diferencia entre minúsculas y mayúsculas.
$resultado = strcasecmp("hola", "HOLA"); //resultado = 0
*Como va al revés de la lógica, se recomienda invertirla.

Título II: Operadores

Concatenación

. : Para concatenar elementos.
.= : Para concatenar a una variable (como el += en otros lenguajes)

Lógicos

&& : AND.
AND : AND.
|| : OR.
OR : OR.
XOR : XOR.
! : NOT.
*Existe una prioridad en estos operadores. Esta prioridad está representada aquí en una tabla, donde el superior es el más prioritario.

Aritméticos

+ : Suma.
++ : Incrementa en uno.
+= : Incrementa en.
- : Resta.
-- : Disminuye en uno.
-= : Disminuye en.
* : Multiplicación.
/ : División.
% : Resto.

Comparación

== : Igual que (no diferencia entre tipos de variable): “5”==5 , true.
=== : Idéntico que (diferencia tipos de variable): “5”===5 , false.
!= : Diferente a.
<> : Diferente a.
< o > : Menor o Mayor que.
<= o >= : Menor o igual o Mayor o igual que.

Título III: Funciones predefinidas

PHP incluye multitud de funciones matemáticas predefinidas y funciones predefinidas:

rand()

Genera un número entero aleatorio.

rand($min, $max)

Genera un número entero aleatorio entre **$min** y **$max**.

pow($base, $exp)

Devuelve **$base** elevado a **$exp** .

round($n)

Redondea **$n** , devolviendo un número entero.

round($n, $precision)

Redondea **$n** , devolviendo un número decimal con **$precision** cifras decimales.

strtolower($string)

Devuelve la cadena **$string** en minúsculas.

strtoupper($string)

Devuelve la cadena **$string** en mayúsculas.

ucwords($string)

Devuelve **$string** con la primera letra de cada palabra en mayúsculas.

header($pagina)

Redirige al usuario a la página indicada en **$pagina**:
header("location:pagina.html");

Refrescar página actual

echo $_SERVER['PHP_SELF'];

date($formato)

Devuelve una cadena de texto formateada con la fecha actual en el formato indicado en **$formato**.
Para elaborar un formato, consultar esto.
date("d/m/Y"); Devuelve día, mes y año actual.
date("d/m/Y H:i:s"); Devuelve día, mes, año, hora, min, sg actual.

empty($var)

Devuelve **true** si la variable (, arreglo u objeto) **$var** está vacía.

isset($var)

Devuelve **true** si la variable **$var** está definida y viceversa.
Podemos indicar dentro el get o el post con el nombre de un botón para ver si este ha sido pulsado o no:
if (isset($_POST["boton"])){}

Título IV: Castings

Implícitos

PHP ofrece castings implícitos, los cuales se realizan de forma automática por el compilador:
$n = "5";

$n += 2;
En este ejemplo, **$n** ha pasado de ser un string a ser un numérico entero.

Explícitos

En algunos casos, nos vemos obligados a realizar castings explícitos, los cuales consisten en poner el tipo entre paréntesis delante de la variable, igual que en Java:

```
$n = 5;
$texto = (int) $n;
```

Título V: Condicionales

if

```
if (condición){
        código;
}else if (condición){
        código;
}else{
        código;
}
```

*Podemos indicar en **condición** una variable booleana sin operador. Si esta será igual a **true** , la condición también y viceversa.
No es necesario especificar siempre **else if y **else**.
*** Podemos indicar tantos **else if** como queramos.

Operador ternario

El operador ternario es una forma de abreviar un **if** en una única línea. El operador ternario devuelve una cosa u otra en función de la condición.
Este operador sigue la siguiente sintaxis:

condición ? returnVerdadero : returnFalso;

```
$edad = 5;
echo $edad >= 18 ? “Eres mayor de edad” : “Eres menor”;
```

switch

```
switch (variable){
        case valor1 :
                código;
                break;
        case valor2 :
                código;
                break;
        default:
}
```

*Podemos sustituir los corchetes por unos dos puntos al principio y la instrucción **endswitch;** al final:

```
switch (variable):
        case valor1 :
                código;
                break;
        default:
endswitch;
```

****default** se ejecutará cuando ningún valor del switch haya servido para la variable, a modo de **else**.
*** No es necesario indicar siempre un default.
**** Los valores de los **case** pueden ser de cualquier tipo de variable, incluso una condición.

Título VI: Bucles

while

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

do while

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

for

```
for($i=valor; condiciónI ; manipularI){
        código;
}
```

foreach

```
foreach($array as $elemento){
        echo $elemento . "<br>";
}
```

En cada vuelta de bucle, cada elemento del arreglo **$array** se pasa a la variable **$elemento** .
(ver Título VIII, Recorrer array asociativo con foreach).

break

Hace que el hilo de ejecución salga del bucle actual.
*Se puede especificar en **while** , **do while** y **for**.

continue

Pasa a la siguiente vuelta del bucle, sin ejecutar más código a partir de esta instrucción.
*Se puede especificar en **while** , **do while** y **for**.

Bucles en HTML

Podemos usar bucles PHP en HTML para repetir un contenido:

```
<?php foreach($array as $elemento): ?>
<p><?php echo $elemento; ?></p>
<?php endforeach; ?>
```

De esta forma, imprimimos los diferentes elementos del array **$array**

Título VII: Funciones

Para crear una función, indicamos **function** , el nombre de la función, unos paréntesis y unos corchetes:

```
function nombre(){
    código;
}
```

Las variables que creamos dentro de las funciones resetean su valor en cada llamada, por lo que a veces es necesario usar variables estáticas, las cuales no resetean su valor.

Para realizar una llamada a la función, especificar su nombre y unos paréntesis:

```
function imprimir(){echo ”Hola”;}
imprimir();
```

Paso de parámetros

Para indicar que una función recibe parámetros, indicar el nombre de una variable entre los paréntesis. Separar los diferentes parámetros por comas:

```
function sumar ($n1, $n2){echo ($n1+$n2);}
sumar(5, 8);            //Salida: 13
```

return

Indica la variable que devuelve la función:

```
function sumar ($n1, $n2){
        return $n1 + $n2;
}
```

return hace que el hilo de ejecución salga de la función:

```
function escribir(){
        return;
        echo “Hola”;
}
```

Es imposible que se imprima Hola .

Parámetros por defecto

Podemos crear un parámetro que no siempre será necesario recibirlo. En caso de que este se reciba, adoptará el valor indicado pero si este no se recibe, adoptará un valor por defecto indicado. De este modo:

```
function convertir ($frase, $mayus=false){
        if ($mayus){
                return strtoupper($frase);
        }else {
                return strtolower($frase);
        }
}

echo convertir(“Hola”);                 //Salida: hola
echo convertir(“Hola”, false);          //Salida: hola
echo convertir(“Hola”, true);           //Salida: HOLA
```

En este caso, si sólo se especifica la frase, devolverá la frase en minúsculas, ya que la variable **$mayus** adopta su valor por defecto. Por el contrario, pasará lo que tenga que pasar según el parámetro.

*Los parámetros por referencia deben ser los últimos que se indiquen en una función.

Parámetros por referencia

En PHP, siempre se pasan los parámetros por valor, por lo que lo que de verdad se pasa en una función es el contenido de la variable.
Podemos especificar que un parámetro será por referencia, por lo que si dentro de la función lo modificamos, estaremos modificando la variable que pasamos , no sólo su valor, por lo que fuera de la función podremos ver los efectos que esta ha tenido sobre la variable.
Para pasar una variable por referencia, indicar un "&" antes del parámetro:
function nombre (&$nombrevar){código;}

Ejemplo por valor:

```
function incrementa ($n){return $n++;}
$num = 5;
echo incrementa($num);   //Salida: 6
echo $num;               //Salida: 5
```

Ejemplo por referencia:

```
function incrementa (&$n){return $n++;}
$num = 5;
echo incrementa($num);   //Salida: 6
echo $num;               //Salida: 6
```

*Los parámetros por defecto se pueden implementar a los parámetros por referencia.

Título VIII: Arreglos

En PHP tenemos dos tipos de arreglos: los arreglos indexados (los que usamos en cualquier lenguaje de programación) o los arreglos asociativos (en formato “clave” => valor).
Los arrays en PHP son dinámicos, por lo que pueden expandirse y encogerse.
Los arrays en PHP pueden almacenar tanto strings como datos numéricos como datos booleanos en el mismo array.

is_array($variable)

Devuelve **true** si **$variable** es un array, **false** si no.

count($array)

Devuelve un int que indica el número de elementos que tiene el arreglo **$array** en su interior.

sort($array)

Ordena los elementos de un array (de menor a mayor o en orden alfabético).

Arrays indexados

Son los arrays que podemos encontrar en cualquier lenguaje de programación.
Para acceder a una posición, indicar el índice entre los corchetes, los cuales comienzan a contar desde 0: **echo $dias[0];**
Si intentamos acceder a una posición no declarada, saltará un error.

Sintaxis 1

Para crear un array indexado, especificamos su nombre (el cual debe empezar por $) seguido de unos corchetes y el valor:

```
$dias [] = “Lunes”;
$dias [] = “Martes”;
$dias [] = “Miércoles”;
```

*Cuando no especificamos un índice en los arrays indexados, estamos haciendo referencia al último elemento +1.

Sintaxis 2

Existe otra sintaxis que se basa en igualar una variable sin corchetes al método **array($valor1, $valor2, ...)** :
$dias = array(“Lunes”, “Martes”, “Miércoles”);

Arrays asociativos

Son arrays que se guardan en formato **“clave”=>valor** .
Para acceder a una posición, indicar el nombre asociativo de cada valor entre los corchetes. **echo $datos[“Nombre”];**
Si intentamos acceder a una posición no declarada, saltará un error.
La clave de un array asociativo no se puede repetir dentro del mismo.

Sintaxis

Para crear un array asociativo, usamos el método **array(“clave”=>valor, “clave”=>valor, ...):**
$datos = array(“Nombre”=>”Juan”, “Apellido”=>”Gómez”, “Edad”=>18);

Agregar datos

Para agregar un nuevo dato a un array asociativo, especificar la nueva clave y su valor:
$datos[“Pais”] = “España”;

Recorrer array asociativo con foreach

Para recorrer un array asociativo con un bucle **foreach** :
$datos = array(“Nombre”=>”Juan”, “Apellido”=>”Gómez”, “Edad”=>18);
foreach($datos as $clave=>$valor){

**echo “A $clave le corresponde $valor
”;**

}

Salida:

A Nombre le corresponde Juan

A Apellido le corresponde Gómez

A Edad le corresponde 18

Arrays multidimensionales

Los arrays multidimensionales son arrays dentro de otros arrays.
Para crear un array de múltiples dimensiones, especificamos un nuevo array dentro de un elemento de un array.
Podemos crear arrays de las dimensiones que queramos.

var_dump($array)

Podemos imprimir un array multidimensional con todos sus elementos con esta instrucción:

echo var_dump($array);

Título IX: Excepciones

Las excepciones son errores que se cometen en tiempo de ejecución.

try catch

Podemos capturar excepciones utilizando un bloque **try catch** , donde en **try** especificamos el código que es susceptible de lanzar excepciones y en catch especificamos el código que ha de ejecutarse si se produce una excepción.
Si ocurre una excepción, no se ejecuta nada del try y se ejecuta el catch, y si no ocurre ninguna, no se ejecuta nada del catch.

La creación de un bloque **try catch** es igual a la de Java:

try{

códigoSusceptibleDeFallar;

}catch (Exception $e){
códigoFallo;
}

finally

Código que se ejecuta siempre tanto si todo va bien como si se produce una excepción.

try{
códigoSusceptibleDeFallar;
}catch (Exception $e){
códigoFallo;
}finally{
códigoSiempre;
}

GetMessage()

Devuelve el texto de error.

Título X: Creación de JSON

JSON es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.

Creamos un array asociativo para posteriormente pasarlo a un JSON con la función **json_encode($array):**

$array = array(“Nombre”=>”Juan”, “Apellido”=>”García”, “Hermanos”=>false);
$json = json_encode($array);

Título XI: Sesiones

Crear una sesión

Para crear sesiones (páginas a las que sólo puedes acceder si te has logueado en algún sitio.), tenemos que ejecutar la función **session_start()** y posteriormente, tenemos que añadir un valor en formato clave valor al array asociativo super global **$_SESSION** donde almacenamos un identificador por ejemplo, el nombre del usuario.
Posteriormente, redirigimos al usuario a la nueva página.

```
session_start();
$_SESSION[“usuario”] = “Pelucho”;
header(“location:menu.php”);
```

Comprobar sesión iniciada

En la página destino, tenemos que crear un código PHP que verifique que la sesión fue iniciada, comprobando que el array superglobal **$_SESSION** dispone del valor que le dimos, comprobando así que el usuario no ha llegado a esta página manipulando la url:
(dentro de menu.php):

```
<?php
	session_start();
	if (!isset($_SESSION[“usuario”])){
		header(“location:login.php”);
	}
?>
```

Dentro del **if** incluimos el código que se ejecutará en el caso de que nuestra sesión no tenga en la variable **$_SESSION** el valor correspondiente a **“usuario”**. De esta forma, también estamos pasando variables entre páginas web.

*Todas las páginas web a las que sólo pueden acceder usuarios registrados, deberán tener esta comprobación.

Cierre de sesión

Por defecto, las sesiones permanecen abiertas, por lo que tenemos que añadir una opción para que cierren sus sesiones.

Para cerrar la sesión actual, podemos incluir un vínculo que nos lleve a una página php donde indicamos la sesión con **session_start()** , destruimos la sesión con **session_destroy()** y finalmente redirigimos la página a otra página, como la de login (eso no lo verá el usuario ya que sucede de forma instantánea):

```
<?php
    session_start();        //Sin esta función, no funciona.
    session_destroy();
    header("location:login.php");
?>
```

Sistema de sesiones en la misma página

1. Para crear una sesión en la misma página, primero tenemos que especificar al botón que verifica el login que tendrá la siguiente instrucción:
echo $_SERVER['PHP_SELF'];
Haciendo esto, refrescamos la página.
Esta instrucción la podemos meter dentro del **action** de un formulario:
action="<?php echo $_SERVER['PHP_SELF']; ?>"
Haciendo esto, recargamos la página pasándole la información del usuario con un método **post** .

2. Para verificar el usuario, tenemos que incluir todo el código correspondiente dentro del código:
if (isset($_POST["nameBoton"])){

}

De esta forma, al pulsar el botón se refrescará la página, comprobando que el botón ha sido pulsado y verificando si existe o no el usuario especificado dentro de nuestro sistema de login.

3. Una vez que sabemos que el usuario existe, iniciamos sesión como vimos en Crear una Sesión. Posteriormente, especificamos a todos los componentes que se mostrarán (o no) si se ha iniciado sesión o no. Por ejemplo, queremos que si no se ha iniciado sesión, se muestra la página login.html con el formulario de inicio de sesión y que si se ha iniciado sesión, se salude al usuario:
(dentro de una parte del body):

```
<?php
    if(!isset($_SESSION[“usuario”])){
        include(“login.html”);
    }else{
        echo “Bienvenido, ” . $_SESSION[“usuario”] . “.”;
    }
?>
```

Título XII: Paso de parámetros entre archivos php

Podemos pasar parámetros entre páginas php usando instrucciones **get** mediante la barra de url.

Envío

Para ello, especificamos en la primera página la apertura de la siguiente página con un método **header()** , donde especificamos una nomenclatura especial en la url:
Si queremos enviar la variable idioma con el valor “es” y la variable usuario con el valor “pelucho” a la página menu.php:

```
header(“location:menu.php?idioma=es&usuario=pelucho”);
```

Especificando un interrogante antes de los parámetros y enviando la información en formato clave=valor, separando los parámetros con &.

Lectura

En la página destino (en este caso, menu.php), podemos acceder al contenido de las variables que pasamos mediante instrucción get, especificando la clave en el array asociativo super global **$_GET** :

$idioma = $_GET[“idioma”]; //”es”
$user = $_GET[“usuario”]; //”pelucho”

Título XIII: AJAX

Podemos comunicarnos con archivos JavaScript usando la tecnología AJAX, en la que JavaScript solicita archivos a php y este se los devuelve.

Leer parámetros

Para rescatar los valores enviados por JavaScript, disponemos de los arrays asociativos super globales **$_GET** y **$_POST** (usar uno u otro dependiendo de la tecnología usada en JavaScript), con los que podemos rescatar los valores especificamos en un json en formato clave:valor, indicando la clave como índice en estos arreglos.

(JSON enviado por **post** en JavaScript):
{user: “Pelucho”, password: 1234}

(lectura en PHP):
$usuario = $_POST[“user”]; //”Pelucho”
$contrasena = $_POST[“password”]; //1234
*Usar **$_GET** si fueron enviados por **get** .

Devolver parámetros

Podemos devolver como parámetros una cadena de texto o un archivo JSON (ver título X).
Para devolver parámetros, simplemente especificar un **echo** delante del parámetro a devolver.
*Como usamos tecnología AJAX, la información no se mostrará en pantalla por el **echo** (ya que la página php nunca se abre), sino que se enviará a la función pertinente en código JavaScript como parámetro.

Ejemplos:
(devolver una cadena de texto):
$txt = “Hola”;
echo $txt;

(devolver un JSON):
$json = json_encode(array(“respuesta”=>true,
“peticion”=>true));
echo $json;

Título XIV: MVC

Los programas PHP deben adoptar una arquitectura de programación llamada MVC (Modelo, Vista, Controlador). Esta arquitectura se basa en separar la parte que genera la interfaz gráfica (Vista) de la parte que maneja los eventos y actualizaciones de la interfaz gráfica (Controlador) y de la parte que contiene la funcionalidad del programa así como las conexiones con BBDD y demás recursos externos (Modelo).
Lo normal en un programa es encontrarnos 3 carpetas: Modelo, Vista, Controlador además de un archivo index que sea el inicio de todo el programa.
Es común que se especifique en el nombre de las clases si pertenecen al modelo, a la vista o al controlador. Ej: **Conexion_modelo**

Título XV: Enviar emails

mail($to, $asunto, $mensaje, $headers)

Devuelve **true** si se ha hecho la intención de enviar el email.
Nos permite enviar emails, especificando en **$to** el destinatario del email, en **$asunto** el asunto del email y en **$mensaje** el cuerpo del email. En **$headers** incluir una serie de expresiones vistas a continuación:

$headers = "MIME-Version: 1.0\r\n";
$headers .= "Content-type: text/html; charset=iso-8859-1\r\n";
$headers .= "From: Nombre < correo >\r\n";
//Nombre: persona que lo envía.
//correo: correo desde el que se envía.

$exito = mail($destinatario, $asunto, $mensaje, $headers);
if ($exito){echo "Correo enviado correctamente"; }

Cada línea se debe separar con "\r\n" como salto de línea.
Las líneas no deben ocupar más de 70 caracteres.

*Para poder enviar emails, debemos tener un servidor de correo electrónico.

Capítulo II: POO

PHP no permite la herencia múltiple, aunque sí permite la múltiple implementación de interfaces, como Java.
En PHP no se usa la nomenclatura del punto.

Título I: Creación de una clase

- Convención: Por convención PHP, la primera letra de los nombres de las clases deben estar en mayúscula.

Para crear una clase, tenemos que seguir la siguiente sintaxis:

```
class nombre{
    código;
}
```

Título II: Creación de campos de clase

Para inicializar los campos de clase, nos situamos dentro de una clase e inicializamos una variable de forma normal, indicando delante la palabra reservada **var** :

```
class nombreClase{
    var nombreCampo;
}
```

Para hacer referencia a un campo de clase dentro de la misma clase, debemos especificar **$this->** antes del nombre del campo.
En cuyo caso, no es necesario especificar el $ de la variable:
$this->nombre;

*Los campos de clase por lo general deben estar encapsulados usando el modificador de acceso **private** (ver Título VII), por motivos de coherencia.

Título III: Creación de métodos

Para crear un método de aclase, simplemente nos situamos dentro de una clase y creamos una función como haríamos para crear una función normal. Estos métodos pueden o no recibir parámetros:

```
class nombreClase{
    function nombreMetodo(parámetros){
        código;
    }
}
```

Para hacer referencia a un método dentro de la misma clase, debemos especificar **$this->** antes del nombre del método.

GETTERS

Los getters son métodos que tienen el objetivo de devolver campos de clase encapsulados:

```
class Coche{
    private $ruedas;
    function get_ruedas(){return $this->ruedas;}
}
```

SETTERS

Los setters son métodos que tienen el objetivo de establecer un valor a un campo de clase encapsulado:

```
class Coche{
    private $ruedas;
    function set_ruedas($ruedas){this->ruedas = $ruedas;}
}
```

Título IV: Creación del método constructor

Para crear un método constructor, nos situamos dentro de una clase y creamos un "método" cuyo nombre sea: **__construct** . Este método puede o no recibir los parámetros que se deseen.

```
class nombreClase{
        function __construct(parámetros){
                código;
        }
}
```

*Para diferenciar los campos de clase de los parámetros del método constructor, especificamos **this->** antes de los campos de clase:

```
class Coche{
        var $ruedas;
        function __construct($ruedas){
                $this->ruedas = $ruedas;
        }
}
```

Título V: Objetos

Creación de instancias / objetos

Para crear instancias de clase (también llamados objetos), tenemos que utilizar la palabra reservada **new** , seguida del nombre de la clase y de unos paréntesis que hacen referencia al método constructor.
Le pasamos parámetros entre paréntesis siempre y cuando el constructor reciba parámetros.
Podemos almacenar las instancias en variables (a las variables se les denomina objetos si no guardan objetos primitivos, como pueden ser instancias de clases).

```
class Coche{}
$coche1 = new Coche();
$coche2 = new Coche();
```

Llamar a métodos

Para llamar a algún método perteneciente a una clase, tenemos que indicar el nombre del objeto, "**->**" y el nombre del método junto con sus paréntesis como si de una función se tratara.

```
class Coche{function saludar () {echo "Hola";}}
$coche = new Coche();
$coche->saludar();          //Salida: Hola
```

Acceder a campos de clase

Para acceder a campos de clase, lo hacemos de forma similar a los métodos:

```
class Coche{
        var $ruedas;
        function __construct ($ruedas){
                $this->ruedas = $ruedas;
        }
}
$coche = new Coche(4);
echo $coche->ruedas;                //Salida: 4
```

Título VI: Herencia

La herencia hace que las clases hijo hereden todos los métodos y campos de clase de clases padre, haciéndolos propios de la hijo.
Para hacer que una clase herede de otra, simplemente debemos especificar **extends** y el nombre de la superclase después del nombre de la clase que estamos creando:

```
class Coche extends Vehiculo{}
```

Sobreescritura de métodos

Para sobreescribir un método (cambiar el comportamiento de un método de una superclase en una subclase), simplemente tenemos que volver a definirlo en la subclase, sin especificar nada nuevo.

En estos casos, es interesante usar método de la superclase (ver Usar métodos de la superclase).

Usar métodos de la superclase / parent

Podemos usar métodos y campos de la superclase especificando **parent::** delante del método o campo de la superclase que queramos usar o acceder:

```
class Vehiculo {
	var $arrancado;
	function __construct(){$this->arrancado = false;}
	function arrancar(){$this->arrancado = true;}
}
class Coche extends Vehiculo{
	código;
	function arrancar(){
		parent::arrancar();
		echo “Coche arrancado”;
	}
}
```

Título VII: Modificadores de acceso

Para especificar un modificador de acceso, especificarlo antes de la declaración del método / campo de clase:

```
private $ruedas;
public function imprimir(){código;}
```

*Cuando especificamos un modificador de acceso a un campo de clase, podemos prescindir de la palabra **var** .

public

Se puede acceder a este método / campo de clase desde fuera de la misma clase.

*Es el modificador por defecto, por lo que si en un campo de clase especificamos sólo **var** , este será **public** y si en un método no especificamos nada, este también será **public**.

private

No se puede acceder a este método / campo de clase desde fuera de la clase.

protected

Igual que **private** a diferencia de que las clases hijo sí que pueden acceder a estos métodos / campos de clase.

Título VIII: Campos y métodos estáticos

Los campos y los métodos estáticos son pertenecientes a la clase, no a los objetos, por lo que debemos usar el nombre de la clase para acceder a estos campos y métodos.
*No podemos usar el operador flecha (->) para acceder a métodos o campos estáticos, sino el operador de dos puntos dobles (::):
nombreClase::$campoEstatico;
nombreClase::metodoEstatico(parámetros);

*Sólo se puede acceder a un campo estático privado desde fuera de la clase con un método estático.

Sintaxis

Para declarar un campo o método estático, indicar **static** antes de su declaración:
static $nombre;
static function nombre(parámetros){código;}

self

Como estos campos y métodos pertenecen a la clase, ya no podemos utilizar el operador **$this->** para acceder desde dentro de la clase, sino el operador **self::** :

```
class Coche{
	static precio;
	function __construct(){self::$precio = 5000;}
}
```

Capítulo III: MySQL

MySQL es un SGDB relacional, multihilo y multiusuario de código libre. Podemos conectarnos con MySQL desde una GUI como phpMyAdmin o desde consola.

- Tipos de datos en MySQL: https://www.youtube.com/watch?v=XJb6qflbsx4&list=PLU8oAlHdN5Bmx-LChV4K3MbHrpZKefNwn&index=16
(ver descripción de vídeo).

Título I: Teoría de conexión a MySQL desde PHP

Para conectarse a una BBDD hace falta conocer 4 cosas:

- Dirección de la BBDD (host).
- Nombre de la BBDD.
- Usuario de la BBDD.
- Contraseña de la BBDD.

Una vez conociendo esto, tenemos que saber que existen varias formas de conectarnos a una BBDD:

- Conexión orientada a objetos:
  - MYSQLi.
  - PDO.
- Conexión por procedimientos.

Pasos

Una vez tengamos claro lo visto, tenemos que seguir los siguientes pasos para realizar toda una conexión:

1. Crear la conexión con la BBDD de forma exitosa.
2. Ejecutar la sentencia SQL deseada almacenando los resultados en un objeto.
3. Recorrer el objeto con los resultados y procesarlos.

4. Cerrar la conexión con la BBDD para liberar recursos.

Título II: Conexión por procedimientos

1. Para conectarnos por procedimientos, tenemos que crear una variable (que será nuestro objeto conexión) con la ayuda del método **mysqli_connect($direccion, $usuario, $contrasena, $nombre)** , a la que le indicamos los 4 parámetros vistos anteriormente (todos estos son datos strings).
*1. La función del punto 1 puede dar bastantes errores, por lo que se recomiendo la siguiente: **mysqli_connect($direccion, $usuario, $contrasena).** Indicaremos el nombre la BBDD entre el punto 1.1. y 1.2. con la instrucción: **mysqli_select_db($conexion, $nombre) or die($mensajeError);**
1.1. Puede que la conexión falle, si esto ocurre, lo podremos saber con un if cuya condición sea la función **mysqli_connect_errno()** , la cual devuelve un **true** si la conexión falló y viceversa.
Es recomendable incluir la función **exit()** para salir del código PHP al final del **if** , para que el punto 2 y 3 no se ejecuten, ya que darán errores.
1.2. Es una buena práctica incluir la función **mysqli_set_charset($conexion, “utf8”);** después de comprobar que la conexión fue exitosa para no tener problemas con la lectura de tildes de la BBDD.

2. Para realizar consultas SQL, utilizamos la función **mysqli_query($conexion, $sql)** donde le indicamos el objeto conexión y la consulta sql que queremos realizar.
Este método lo podemos igualar a una variable (que adoptará el valor de un objeto **resultset**), que almacenará los datos de la consulta (en caso de que esta sea una consulta SELECT).

*2. Si no vamos a realizar una consulta select, sino una de acción, la variable que devuelve **mysqli_query()** será **true** si la consulta se realizó con éxito y **false** si no.
En este caso, no es necesario hacer nada del punto 3.
*2.1. Disponemos de la función **mysqli_affected_rows($conexion)** que nos devuelve un int con el total de registros afectados con un query de tipo CRUD

3. Para ver los resultados del **resultset** , tenemos que ir fila por fila con la función **mysqli_fetch_row($resultset)** , que nos devuelve un array indexado, que podemos guardar en una variable, con el contenido de la primera fila de la consulta y avanza a la siguiente fila, por lo que si la volvemos a ejecutar, leeremos la segunda fila.
*3. Podemos utilizar la función **mysqli_fetch_assoc($resultset)** que funciona igual que **mysqli_fetch_row()** , salvo que en vez de ser un array indexado, es un array asociativo, cuyas claves son los nombres de cada campo de la tabla.
3. También podemos usar la función **mysqli_fetch_all($reultset) , que nos devuelve un array de arrays con el resultado completo de la consulta.
3.1. Para hacer esto dentro de un bucle, creamos un bucle **while** cuya condición sea que el proceso de obtener una fila se ha realizado con éxito (**==true**):

4. Para terminar, cerramos la conexión para liberar recursos con la función **mysqli_close($conexion)** .

$direccion = “DireccionBBDD”;
$usuario = “UsuarioBBDD”;
$contrasena = “ContrasenaBBDD”;
$nombre = “Pruebas”; //Sustituir por el nombre en cuestión.

$conexion = mysqli_connect($direccion, $usuario, $contrasena);

```
if(mysqli_connect_errno()){
	echo “La conexión con la BBDD ha fallado”;
	exit();
}
mysqli_select_db($conexion, $nombre) or die(“Algo ha fallado”);
mysqli_set_charset($conexion, “utf8”);

$resultado = mysqli_query($conexion, “select * from Productos”);

while($fila = mysqli_fetch_row($resultado)){
	foreach ($fila as $elemento){
		echo $elemento . “ ”;
	}
	echo “<br>”;
}

/*while($fila = mysqli_fetch_assoc($resultado)){
	foreach ($fila as $clave=>$valor){
		echo “$clave: $valor , ”;
	}
	echo “<br>”;
}*/

mysqli_close($conexion);
```

Título III: Consultas preparadas por procedimientos

Las consultas preparadas son consultas prefijadas que nos permiten realizar consultas de una forma más eficiente y segura, previniendo la inyección SQL.

1. Establecemos la conexión con la BBDD (ver título II).

2. Preparamos una consulta con la función **mysqli_prepare($conexion, $sql)** , indicando nuestro objeto conexión y la consulta sql, sustituyendo los parámetros de búsqueda por un interrogante.
Esta función devuelve un objeto **Mysqli_stmt** , por lo que debemos almacenarlo en una variable.

3. Unimos los parámetros que queremos enviar con la sentencia con la función **mysqli_stmt_bind_param($tipoDato, $dato, $mysqli_stmt)** , donde especificamos la sentencia preparada, el tipo de dato que queremos enviar (entero = “i”, decimal = “d”, string = “s” (*)) y el dato que enviamos.
Devuelve **true** si todo se ha realizado correctamente, **false** si no.
*Si indicamos en **$sql** más de 1 ? , especificar el tipo de dato así:
3 ? : “sss” | 7 ? : “sisssss” (depende de los tipos de dato).

4. Una vez creada la consulta preparada, la podemos ejecutar cuando queramos usando la función **mysqli_stmt_execute($mysqli_stmt)** , la cual devuelve **true** si todo se ha realizado correctamente, **false** si no.
*4. Si nuestra sentencia es de tipo create, update o delete, ignorar pasos 5 y 6, ya que no recibiremos nada.

5. Asociamos variables al resultado de la consulta con la función **mysqli_stmt_bind_result($mysqli_stmt, $var1, $var2, ...)**. Le indicamos el objeto **Mysqli_stmt** y las variables en las que almacenará los resultados. Indicar tantas variables como campos tenga el resultado.

6. Leemos los valores con la función **mysqli_stmt_fetch($mysqli_stmt)** .

7. Una vez terminadas todas las consultas, cerramos la conexión con **mysqli_close($conexion);**

(Queremos seleccionar todos los campos del id=5)

```
$conexion = mysqli_connect($direccion, $usuario, $contrasena);
if(mysqli_connect_errno()){
        echo “La conexión con la BBDD ha fallado”;
        exit();
}
mysqli_select_db($conexion, $nombre) or die(“Algo ha fallado”);
mysqli_set_charset($conexion, “utf8”);

$mysqli_stmt = mysqli_prepare($conexion, “select * from tabla
where id = ?”);

$ok = mysqli_stmt_bind_param($mysqli_stmt, “i”, 5);
if(!$ok){echo “Algo ha fallado añadiendo el parámetro”; exit();}

$ok = mysqli_stmt_execute($mysqli_stmt);
if(!$ok){echo “Algo ha fallado en la consulta”; exit();}

$ok = mysqli_stmt_bind_result($mysqli_stmt, $campo1, $campo2,
$campo3, $campo4);

while(mysqli_stmt_fetch($mysqli_stmt)){
        echo $campo1 . “ “ . $campo2 . “ “ . $campo3 . “ “. $campo4;
}

mysqli_close($conexion);
```

Título IV: Prevención de inyecciones SQL

Las inyecciones SQL consisten en inyectar un código SQL adicional que se agrega a nuestras instrucciones SQL, de forma que se pueden manipular para acceder a contenido de forma malintencionada.
Por todo esto, existen algunos procedimientos que podemos seguir para prevenir que seamos atacados por inyecciones SQL:

mysqli_real_escape_string($conexion, $string)

En la mayoría de los casos, los datos que introducimos en consultas SQL en cláusulas WHERE son variables de tipo String.
Esta función devuelve el string indicado en **$string** , ignorando caracteres especiales como **‘** o **&** , para prevenir inyecciones SQL.
*Recomendable especificar esta función en todas las variables que incluyamos dentro de sentencias SQL.
**Sólo útil en conexión por procedimientos.

addslashes($string)

Realiza la misma función que **mysqli_real_escape_string()** pero de forma simplificada.
*Se recomienda el uso de **mysqli_real_escape_string()** y no el de **addslashes()** .

Título V: Conexión orientada a objetos: MYSQLi

Este estilo de conexión es el más moderno y recomendable.
Las conexiones orientadas a objetos se basan en la creación de un objeto y en la ejecución de métodos sobre él.

1. Para crear la conexión, nos creamos una instancia de la clase **mysqli** con el constructor **($direccion, $usuario, $contrasena, $nombre)** , donde indicamos todos los parámetros de la BBDD.
1.1. Dentro de un **if** , miramos si el campo de clase **connect_errno** es igual a **true** . Si esto es cierto, habrá habido un fallo en la conexión.

1.2. Especificamos la codificación de la BBDD con el método **set_charset($codificacion)** , al cual le indicaremos por lo general **“utf8”**.

2. Ejecutamos el método **query($sql)** sobre nuestro objeto conexión con la sentencia SQL a ejecutar.
Este método devuelve un objeto con el resultado de la sentencia en caso de que fuese de tipo SELECT.
*2. Si la sentencia SQL era de creación, actualización o eliminación, no es necesario almacenar el objeto, ya que no devolverá nada. También podemos ignorar el punto 3.
2.1. Si la ejecución del método falló, el campo de clase **errno** de nuestro objeto conexión será igual a **true** .

3. Recorremos el resultado con un bucle utilizando el método **fetch_assoc()** sobre el objeto que obtuvimos en el punto 3, el cual nos devuelve un array asociativo de la primera fila. Cada vez que este se ejecuta, pasa a la siguiente fila.
*3. También podemos usar el método **fetch_array()** para un array indexado.
3. También podemos usar el método **fetch_all() , que nos devuelve un array de arrays asociativos con el resultado ENTERO de la consulta.

4. Cerramos la conexión con la BBDD con el método **close()**.

```
$conexion = new mysqli($direccion, $usuario, $contrasena, $nombre);
if($conexion->connect_errno){
        echo “La conexión ha fallado”;
        exit();
}
$conexion->set_charset(“utf8”);
```

```
$resultado = $conexion->query("select * from tabla");
if($conexion->errno){echo "Algo ha fallado"; exit();}

while($fila = $resultado->fetch_assoc()){
        foreach ($fila as $valor){
                echo "$valor ";
        }
        echo "<br>";
}

/*$consulta = $resultado->fetch_all();
foreach ($consulta as $fila){
        foreach($fila as $valor){
                echo $valor . " ";
        }
}*/

$conexion->close();
```

Título VI: Conexión orientada a objetos: PDO

PDO (PHP Data Object) es una librería que nos permite conectarnos con BBDD. Este código se sitúa entre nuestro código PHP y las BBDD.
PDO nos permite conectarnos con multitud de SGDB .
Podemos consultar todos los métodos de PDO aquí.

Dentro de un **try catch** :
1. Para crear la conexión, creamos una instancia de la clase PDO , usando el constructor **($direccion, $usuario, $contrasena)**.
1.1. Añadimos la siguiente línea de código para que en caso de haber un error, se lance una excepción.

1.2. Especificamos la codificación de la base con el método **exec($sentencia)** , al que le indicamos como sentencia: **“SET CHARACTER SET codificado”**.
*En dirección , tenemos que usar una sintaxis específica, en la que especificamos el tipo de SGDB , host y nombre de la BBDD: **“mysql:host=direccion; dbname=nombre”**
** Este código puede lanzar una excepción si falla la conexión, por lo que debemos introducirlo dentro de un **try catch**. Dentro de un **finally** podemos igualar este objeto a **null** para liberar recursos.

2. Ejecutamos la sentencia SQL con el método **query($sql)** sobre el objeto conexión, para ejecutar la consulta.
Este método devuelve un objeto **PDOStatement** con el resultado de la query o un **false** en caso de que se haya producido algún error.

3. Usamos el método **fetchAll($tipoArray)** sobre el objeto **PDOStatement**, que nos devuelve un array de arrays asociativos con la consulta ENTERA de la sentencia SQL.
Tipos de arreglos:

- **PDO::FETCH_ASSOC** para un array asociativo.
- **PDO::FETCH_NUM** para un array indexado.

```
try{

$base = new PDO(“mysql:host=direccion; dbname=nombre”, $usuario, $constrasena);
&base->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$base->exec(“SET CHARACTER SET utf8”);

$resultado = $base->query(“select * from tabla”);

$array = $resultado->fetchAll();
```

```
foreach($array as $fila){
        foreach($fila as $clave=>$valor){
                echo $clave . ": " . $valor . " ";
        }
        echo "<br>";
}

}catch (Exception $e){
        echo "Ha habido un error";
}finally {
        $base = null;
}
```

Título VII: Consultas preparadas con PDO

Las consultas preparadas son consultas prefijadas que nos permiten realizar consultas de una forma más eficiente y segura, previniendo la inyección SQL.

Dentro de un **try catch** :
1. Primero, tenemos que crear la conexión con la BBDD (ver título VI).

2. Preparamos la consulta preparada con el método **prepare($sql)** sobre nuestro objeto conexión, indicando interrogantes por las variables.
Este método nos devuelve un objeto llamado **PDOStatement**.

3. Ejecutamos la consulta preparada con el método **execute($array)** sobre nuestro objeto **PDOStatement** . En **$array** le indicamos un arreglo indexado con el valor de las variables que sustituirán a los interrogantes (ver Marcadores).

*Si nuestra consulta preparada no tiene ningún interrogante o marcador, especificar un array vacío: **array()**

4. Pasamos el resultado de la consulta a variables con el método **fetch($tipoArray)** sobre el objeto **PDOStatement**. Donde le indicamos en **$tipoArray** el tipo de array que devolverá:

- **PDO::FETCH_ASSOC** para un array asociativo.
- **PDO::FETCH_NUM** para un array indexado.

*Esto no es necesario en consultas create, update o delete.
*4. También podemos usar el método **fetchAll($tipoArray)** sobre el objeto **PDOStatement**, que nos devuelve un array de arrays asociativos con la consulta ENTERA de la sentencia SQL.
4. Disponemos del método **rowCount() del objeto **PDOStatement** que nos devuelve el número de filas que ha tenido la consulta de nuestra sentencia como un entero.

5. Cerramos el objeto **PDOStatement** para liberar recursos ejecutando sobre él el método **closeCursor()**.

6. Cerramos la conexión especificando que la conexión será igual a **null** dentro de un **finally** .

(Queremos seleccionar todos los campos del id=5)

```
try{

$base = new PDO("mysql:host=direccion; dbname=nombre",
$usuario, $constrasena);
&base->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
$base->exec("SET CHARACTER SET utf8");

$resultado = $base->prepare("select * from tabla where id = ?");
```

```
$resultado->execute(array(5));

while($fila = $resultado->fetch(PDO::FETCH_ASSOC)){
        foreach($fila as $clave=>$valor){
                echo $clave . ": " . $valor . " ";
        }
        echo "<br>";
}

$resultado->closeCursor();

}catch (Exception $e){
        echo "Ha habido un error";
}finally {
        $base = null;
}
```

Marcadores

Los marcadores nos permiten sustituir interrogantes por un nombre para poder usar arrays asociativos en el método **execute()**. Las claves del array asociativo serán los marcadores. Un marcador se construye con dos puntos seguidos de un nombre, como si creamos una variable:

Queremos seleccionar todos los campos de una tabla donde el id sea igual a 5 o la seccion sea igual a deportes o el país sea igual a España:

(forma SIN marcadores)

```
$sql = "select * from tabla where id = ? or seccion = ? or pais=?";
$resultado = $base->prepare($sql);

$array = array(5, "deportes", "España");
$resultado->execute($array);
```

(forma CON marcadores)

$sql = “select * from tabla where id = :id or seccion = :sec or pais = :pais ”;
$resultado = $base->prepare($sql);

$array = array(“:id”=>5, “:sec”=>”deportes”,
“:pais”=>”España”);
$resultado->execute($array);

*También podemos añadir los valores a la sentencia y luego ejecutarla:
$resultado->bindValue(“:id”, 5);
$resultado->bindValue(“:sec”, “deportes”);
$resultado->bindValue(“:pais”, “España”);
$resultado->execute();

Título VIII: Encriptación de información

Podemos encriptar información como contraseñas con funciones de php para almacenarlas de forma encriptada en nuestras BBDD.

password_hash($informacion, $sal)

Para ello, usamos la función **password_hash($informacion, PASSWORD_DEFAULT)** , donde especificamos en **$informacion** la información a encriptar, como una contraseña.
Este método nos devuelve la información encriptada.

$contraEnc = password_hash($contra, PASSWORD_DEFAULT);

*La información encriptada puede ser muy larga, así que en la BBDD especificaremos un varchar de 256 caracteres, aunque realmente nuestras contraseñas sin encriptar sean de menos de 20 caracteres.
**El tiempo de cifrado de una contraseña corta es el mismo para una contraseña larga.

password_hash($informacion, $sal, $coste)

Podemos ampliar la seguridad de nuestro cifrado ampliando el coste del encriptado. De esta forma, nuestro encriptado tardará más tiempo pero será más difícil desencriptarlo (aunque el coste default ya es bastante seguro).

Para indicar este parámetro, especificar un array asociativo con la clave cost y un valor correspondiente al coste (10 es el valor default, no subir mucho más de 15).

```
$modo = PASSWORD_DEFAULT;
$contraE = password_hash($contra, $modo, array("cost"=>12));
```

password_verify($cotrasena, $hash)

Ya que las contraseñas de la BBDD están cifradas, no podemos compararlas con las que introduce el usuario sin cifrar de otra forma diferente a la función **password_verify()** , la cual admite dos parámetros: la contraseña sin cifrar que introduce el usuario (**$contrasena**) y la contraseña cifrada de nuestra BBDD (**$hash**). Devuelve **true** si ambas coinciden, **false** si no.

```
if (password_verify($contra, $contraE)){
        echo "Contraseña correcta";
}
```

Capítulo IV: Cookies

Qué es una Cookie

Todos los navegadores tienen un pequeño espacio en el disco duro dedicado a las Cookies.
Las Cookies son ficheros de texto plano que son capaces de almacenar diversos elementos, como el nombre del usuario o el carro de la compra en algunas aplicaciones.
El programador puede definir cuánto tiempo permanecerán las cookies en el ordenador del usuario.

Un ejemplo de utilidad de las cookies es almacenar los sitios web donde te metes para mostrarte publicidad relacionada.

Las cookies también nos permiten transmitir datos entre páginas web, recordar el idioma en el que quieres ver la página, recordar usuarios y contraseñas...

*Por curiosidad, se le aplica el término “cookie” por el cuento de Hansel y Gretel en el que van soltando galletas para marcar el camino, como si estuviésemos registrando los sitios por los que hemos pasado (o navegado).

Título I: Creación de cookies

Para crear una cookie, ejecutamos el método **setcookie($clave, $valor)** , al que le pasamos dos parámetros con la información que contendrá la cookie en un formato de clave-valor:
setcookie(“user”, “Pelucho”);

Tiempo de vida

Al no especificar un tiempo de vida, como en el ejemplo superior, las cookies se eliminan al cerrar el navegador.

Para que esto no suceda y que podamos cerrar el navegador sin borrar la cookie, añadimos un tercer parámetro a la función especificando la función **time()** (que recibe el tiempo actual) y le sumamos los segundos que se corresponderá con el tiempo de vida de la cookie:

setcookie($clave, $valor, time()+30); //La cookie dura 30 seg.

setcookie($clave, $valor, time()+300); //La cookie dura 5 min.

*Aparentemente, no existe un máximo de tiempo de vida.

Directorios de lectura

Por defecto, la cookie es accesible desde cualquier archivo php que se encuentre en el mismo directorio donde se ha creado la cookie o en todos los subdirectorios del directorio actual.

Para cambiar esto, especificar un cuarto parámetro donde especificamos la ruta de la carpeta en la que sólo podremos leer la cookie entre comillas.

Usar como raíz la raíz de nuestro servidor.

setcookie($clave, $valor, $tiempo, "/pagina/php/")

Nuestra cookie sólo será accesible desde dentro de la carpeta php que a su vez está dentro de la carpeta pagina.

Título II: Lectura de cookies

Para leer una cookie, utilizamos el array asociativo super global **$_COOKIE** en el que se almacenan todas las cookies en formato clave-valor:

echo $_COOKIE["user"]; //Salida: Pelucho

Comprobación de cookies

Si la cookie no existe, nos arrojará un error. Para que esto no suceda, primero debemos comprobar que la cookie existe y, en caso afirmativo, leer la cookie:

```
if ( isset($_COOKIE["user"])){
        echo $_COOKIE["user"];          //Salida: Pelucho
```

}

*También podemos especificar la propia cookie como condición del **if**
if ($_COOKIET[“user”]){
echo $_COOKIE[“user”];
}
Aunque se recomienda la primera forma.

Título III: Eliminación de cookies

Para eliminar una cookie desde PHP, copiamos el método que usamos para crear la cookie y le especificamos un tiempo negativo:
Si queremos eliminar la cookie:
setcookie($clave, $valor, time()+300);

Ejecutamos la siguiente función: **setcookie($clave, $valor, time()-1);**
Especificando los mismos **$clave** , **$valor** (y posibles parámetros extra) que especificamos para crearla.

Capítulo V: Imágenes

Esta parte del curso fue saltada (vídeos 83 - 87).