UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

Programador Web Inicial

www.sceu.frba.utn.edu.ar/e-learning

Módulo 3:

Programación Web (parte 2)

Unidad 3:

Vectores y funciones

Presentación de la Unidad:

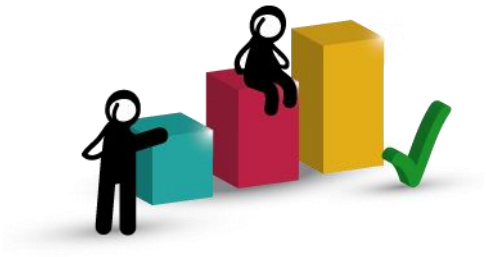
Vamos a ver varios ejemplos de trabajo con arrays (vamos a poder encontrarlos también como arreglos, vectores, matrices o tablas en castellano) en PHP que ilustrarán un poco el funcionamiento de algunas de las funciones de arrays más populares que trae consigo PHP.
El concepto de función podría ser definido como un conjunto de instrucciones que permiten procesar las variables para obtener un resultado.

Objetivos:

- Aprender cuales son las alternativas de configuración del lenguaje PHP.
- Aprender como es su funcionamiento.
- Aprender la sintaxis del mismo y como se intercalan las instrucciones dentro del código HTML

BLOQUES TEMÁTICOS:

.: Vectores (array)
    .: Array indexado
    .: Array asociativo
.: Funciones en PHP
    .: Funciones y procedimientos
.: Funciones del lenguaje
    .: Función mail()
.: Incluyendo archivos
    .: Include()
.: Funciones en PHP para MySQL
    .: mysql_connect()
    .: mysql_select_db()
    .: mysql_query()
    .: mysql_free_result()
    .: mysql_close()
    .: mysql_create_db()
    .: mysql_fetch_row()
    .: mysql_fetch_array()

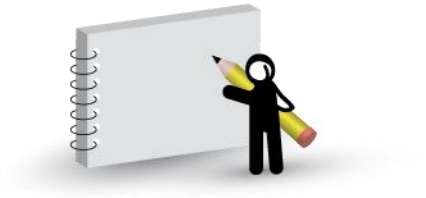


CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen las actividades sugeridas y compartan en los foros los resultados obtenidos.



TOMEN NOTA*

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

VECTORES (ARRAY)

Los arreglos pueden ser de una o más dimensiones, los de una dimensión, son llamados comúnmente "vectores".

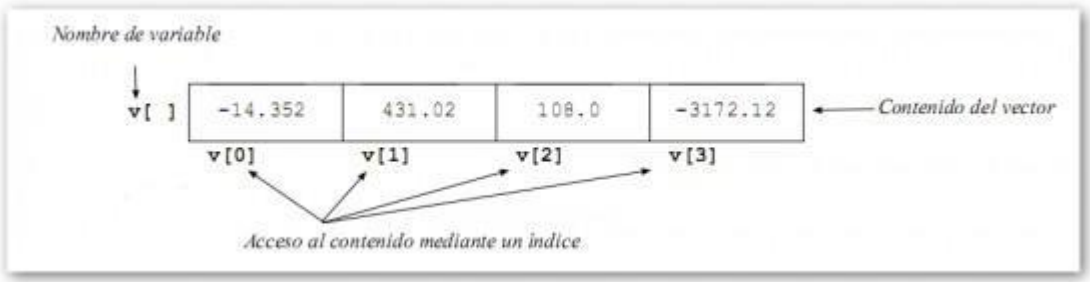
A diferencia de otros lenguajes, en PHP, un vector o array puede tener elementos de distintos tipos.

Arrays indexados

Las variables de tipo Array son como variables con muchos "**compartimientos**" que pueden almacenar varios valores, a los que se puede acceder mediante un índice. Es decir, un **array** es una variable que está compuesta de varios elementos, cada uno de ellos catalogado dentro de ella misma por medio de una clave.

Dijimos entonces que para hacer referencia a un elemento del vector, se utiliza un índice, que indica la dirección en donde se encuentra un determinado valor. El índice en un arreglo comienza siempre por cero. (Más adelante veremos que el índice de un vector, no necesariamente debe ser un número entero, sino que también puede ser un texto).

En la sintaxis de los Arrays en PHP, el índice (o clave) se indica entre corchetes.



Hay varias maneras de definir un array:

a) Dándole un valor a cada posición accediendo individualmente por su índice:

```
<?php

$familia[0]="Padre";
$familia[1]="Madre";
$familia[2]="Hijo";

echo $familia[0]; //Muestra 'Padre'
echo $familia[1]; //Muestra 'Madre'
```

```
?>
```

Otra forma de inicializar un vector, es a través del constructor array, es decir, definiendo los valores iniciales separados por comas dentro de array( ), como se muestra en los siguientse ejemplos:

```
<?php
$pais =array("Argentina","Uruguay","Brasil","Chile");
$familia = array("Padre","Madre","Hijo");

echo $familia[0]; //Muestra 'Padre'
echo $pais[1]; //Muestra Uruguay
?>
```

Los índices de los array **siempre empiezan desde 0**, así que hay que tener en cuenta que si se quiere acceder al segundo elemento contenido en un array, habrá que acceder a él mediante el índice 1, por ejemplo.

Existen otras maneras de inicializar vectores en PHP:

```
$Pais[] = "Argentina";
$Pais[] = "Uruguay";
$Pais[] = "Brasil";
$Pais[] = "Chile";
```

En este caso se observa que no es necesario colocar el número de índice, ya que PHP lo asigna automáticamente para cada valor, comenzando siempre desde cero.

También se puede definir un arreglo asociando explícitamente el índice a un valor, como se indica a continuación:

```
$Frutas = array(0 => "Manzana",
1 => "Naranja",
2 => "Pera",
3 => "Ananá");
```

Además, los índices, pueden no ser obligatoriamente consecutivos, ni tampoco comenzar de cero, ni tampoco ser un número.

Se puede conocer la cantidad de elementos que tiene un vector, para ello se utiliza la función **count(vector)**. Esta función acepta como parámetro el nombre del vector y devuelve la cantidad de elementos del mismo.

Ejemplo:

Almacenar los nombres de los días de la semana en un vector y luego imprimirlos uno debajo de otro.

```
<html>
<title> ejmeplo 1 </title>
<body>
<?php
// inicializacion del vector
$dia[0] = "domingo";
$dia[1] = "lunes";
$dia[2] = "martes";
$dia[3] = "miércoles";
$dia[4] = "jueves";
$dia[5] = "viernes";
$dia[6] = "sábado";
// impresion del vector
for($i=0; $i<7; $i++)
{
echo ($dia[$i] . "<br/>") ;
}
?>
</body>
</html>
```

Se inicializa el vector indicando el número que le corresponde a cada posición entre corchetes [ ] y asignando el valor que se desea almacenar en dicha posición.

Un vector, en PHP, puede contener elementos de distintos tipos de datos, es decir, un elemento puede ser un número entero, otro una cadena, otro un número con decimales, etc. Un modelo de este caso se puede observar en el siguiente ejemplo.

Ejemplo:

Almacenar en un vector los datos personales de un empleado y luego mostrarlos en pantalla.

```
<html>
<title> ejemplo 2 </title>
<body>
<?php
// inicializacion del vector
$empleado[0] = 4371;
$empleado[1] = "martinez leandro";
$empleado[2] = "27.643.742";
$empleado[3] = 1429.54;
```

```
$empleado[4] = "arquitecto";
// impresion del vector
echo ("legajo: " . $empleado[0] . "<br/>");
echo ("nombre: " . $empleado[1] . "<br/>");
echo ("dni : " . $empleado[2] . "<br/>");
echo ("sueldo: " . $empleado[3] . "<br/>");
echo ("profesion: " . $empleado[4] . "<br/>");
?>
</body>
</html>
```

Ejemplo:

Cargar en un vector artículos de librería y luego imprimir la cantidad de ellos.

```
<html>
<title> ejemplo 3 </title>
<body>
<?php
// inicializacion del vector
$articulos =array("lápis","goma","hoja","tinta");
// impresion del vector
$cantidad = count($articulos);
echo ("la cantidad de artículos son: " . $cantidad);
?>
</body>
</html>
```

Arrays asociativos

Hemos visto anteriormente lo que son los arrays y a como operar con ellos de forma muy elemental. Hemos aprendido que los arrays se asignan a variables.

Estas variables no tienen asignados valores, sino elementos de array que son datos que están asociados, a su vez, a un elemento del array llamado índice.

Este índice se caracteriza por conectar los elementos del array por medio de una numeración que empieza por cero. Así, el primer elemento del array tiene índice cero, el segundo tiene índice uno, y así sucesivamente.

Pero en realidad, resulta que este índice es numérico solo por defecto; es decir, tenemos la posibilidad de crear nuestro propio índice dentro de un array. Cuando hacemos esto, estamos convirtiendo el array en un array asociativo.

```
<?php
$menu = array(
```

```
"Primer plato" => "Rabas",
"Segundo plato" => "Pejerrey al verdeo con papas rústicas",
"Postre" => "Helado"
);
echo $menu["Primer plato"];
?>
```

Si repasamos el código anterior, hemos asignado a la variable `$menu` un array asociativo. Al ser asociativo tenemos que especificar un índice. El índice que hemos especificado es: *Primer plato*, *Segundo plato* y *Postre*. Posteriormente, hemos realizado una sentencia `echo` para sacar en pantalla el primer plato del menú.

En el paréntesis del array, tenemos que asociar el índice con el valor por medio del operador `=>`.

Después, para acceder a un elemento del array asociativo tenemos que escribir la variable asignada al array, y posteriormente escribir entre corchetes el índice que hemos asignado a dicho elemento del array.

Tenemos la posibilidad de poner, para especificar el índice, cualquier tipo de dato. Por defecto es un número, pero también podemos poner cadenas de texto, e incluso variables o funciones.

```
<?php
$indice = "favorito";
$color = array($indice => "violeta");
echo $color[$indice];
?>
```

Este concepto de asociar datos a elementos del array por medio del índice nos permitirá dar el siguiente paso en el tema de los arrays.

Resumiendo entonces, la diferencia de los Arrays asociativos con los indexados, es que en vez de acceder al contenido de un "compartimiento" del array por el índice de su posición, podemos definir las CLAVES con cadenas que identifiquen mejor el contenido de cada uno de esos compartimientos.

Al tener cadenas en vez de un número como clave, es mucho más fácil identificar el valor de cada elemento del array:

```
<?php
$precios["TV"]=700;
$precios["Telefono"]=150;
$precios["Computadora"]=1900;
echo $precios["Computadora"]; //Muestra '1900'
?>
```

Otra manera de definir un array asociativo es:

```
<?php
$precios=array("TV"=>1500, "Telefono"=>150, "Computadora"=> 1900);
echo $precios["Computadora"]; //Muestra '1900'
?>
```

Observación: una función muy práctica para usar con arrays es count, que devuelve la cantidad de elementos de un array.

```
<?php
$un_array=array(4,6,3,6,7,23,1);
echo count($un_array); //Muestra “7”
?>
```

Como ya sabemos, un array sencillo está formado por conjuntos de parejas índice => valor, o como suele expresarse en inglés, key, value. Hasta ahora hemos manejado un ejemplo con índices o keys numéricos (también conocidos como arrays escalares), pero también podemos usar strings como índices, es decir, cadenas de texto. Este tipo de array es el array asociativo:

```
<?php
$mis_barrios = array(
"norte"=>"Vicente Lopez",
"sur"=>"Avellaneda",
"oeste"=>"Castelar");
?>
```

FUNCIONES EN PHP

Funciones y procedimientos

El concepto de función podría ser definido como un conjunto de instrucciones que permiten procesar las variables para obtener un resultado. Puede que esta definición resulte un poco vaga si no nos servimos de un ejemplo para ilustrarla.

Supongamos que queremos calcular el valor total de un pedido a partir de la simple suma de los precios de cada uno de los artículos.

Tendríamos que seguir el siguiente razonamiento:

definir función suma(art1,art2,art3)
suma=art1+art2+art3
imprimir(suma)
fin función

Este supuesto programa nos permitiría calcular la suma de tres elementos e imprimir el resultado en pantalla. Lo interesante de utilizar este tipo de funciones es que ellas nos permiten su utilización sistemática tantas veces como queramos sin necesidad de escribir las instrucciones tantas veces como veces queremos utilizarla. Por supuesto, podemos prescindir de esta declaración de función e introducir una línea del siguiente tipo:

imprimir (art1+art2+art3)

Evidentemente, cuanto más complicada sea la función y más a menudo la utilicemos en nuestros scripts más útil resulta definirlas.

Esta función suma podría ser utilizada en cualquier lugar de nuestro script haciendo una llamada del siguiente tipo:

ejecuta suma(4,6,9)

Cuyo resultado sería:

19

Del mismo modo, los procedimientos son parecidos a las funciones. La diferencia consiste tan solo en que en estos últimos el interés no radica en el resultado obtenido sino más bien en las operaciones realizadas al ejecutarla (creación de un archivo, reenvío a otra página,...).

En lenguajes como el PHP las funciones y los procedimientos son considerados como la misma cosa y para definirlos se hace usando los mismos comandos.

Tanto las variables como las funciones y los procedimientos deben ser nombradas sin servirse de acentos, espacios ni caracteres especiales para no correr riesgos de error.

Existen en PHP una gran variedad de funciones que ya vienen programadas, no obstante nos permite la creación de nuestras propias funciones, veamos ambos casos.

FUNCIONES DEL LENGUAJE

Desarrollaremos a continuación algunas de las funciones más utilizadas.

Función mail()

Para el envío de correos electrónicos utilizando PHP disponemos de una función bastante potente, incluida en todas las versiones de PHP, que nos permite mandar mails sin necesidad de instalar ningún añadido, distinto a lo que nos permitía HTML quien necesita de un programa de correo instalado en la correspondiente máquina.

En concreto, en PHP disponemos de una función llamada ***mail()*** que permite configurar y enviar el mensaje de correo. La función recibe tres parámetros de manera obligatoria y otro parámetro que podemos colocar opcionalmente. Devuelve true si se envió el mensaje correctamente y false en caso contrario.

Parámetros necesarios en todos los casos

Destinatario: la dirección de correo o direcciones de correo que han de recibir el mensaje. Si incluimos varias direcciones debemos separarlas por una coma.

Asunto: para indicar una cadena de caracteres que queremos que sea el asunto del correo electrónico a enviar.

Cuerpo: el cuerpo del mensaje, lo que queremos que tenga escrito el correo.

Ejemplo:

```
<?php
mail("lorena.bernis@gmail.com", "Este es el asunto del mail", "Este es el cuerpo del mensaje");
?>
```

Parámetros opcionales del envío de correo

Headers: Cabeceras del correo. Datos como la dirección de respuesta, las posibles direcciones que recibirán copia del mensaje, las direcciones que recibirán copia oculta, si el correo está en formato HTML, etc.

Ejemplo complejo de envío de correo:

Vamos a enviar un correo con formato HTML a Lorena.bernis@gmail.com, con copia a lorenabernis@hotmail.com.ar y con copia oculta para lbernis@yahoo.com.ar y lorena@ideasenpixel.com.ar

La dirección de respuesta la configuraremos a lorena@gmail.com

```
<?php
$destinatario = "lorena.bernis@gmail.com";
$asunto = "Este mensaje es una prueba";
$cuerpo = '
<html>
<head>
<title>Prueba de correo</title>
</head>
<body>
<h1>Hola amigos!</h1>
<p>
<strong>Bienvenidos a este correo electrónico de prueba</strong>. Estamos haciendo una prueba del funcionamiento de mail(). Este cuerpo del mensaje corresponde a la prueba de envío de mails por PHP. Habría que cambiarlo para poner tu propio cuerpo. De igual manera, debería cambiarse también la cabecera del mensaje.
</p>
</body>
</html>
';
//para el envío en formato HTML
$headers = "MIME-Version: 1.0\r\n";
$headers .= "Content-type: text/html; charset=iso-8859-1\r\n";
//dirección del remitente
$headers .= "From: Lorena Bernis < lorena.bernis@gmail.com >\r\n";
//dirección de respuesta, si queremos que sea distinta que la del remitente
$headers .= "Reply-To: lorena@gmail.com.ar\r\n";
//direcciones que recibián copia
$headers .= "Cc: lorenabernis@hotmail.com \r\n";
//direcciones que recibirán copia oculta
$headers .= "Bcc: lbernis@yahoo.com.ar, lorena@ideasenpixel.com.ar \r\n";
```

```
mail($destinatario,$asunto,$cuerpo,$headers);
?>
```

Nota: Antes de poner en marcha el script en el servidor, por favor,

cambien los datos de configuración de las direcciones de correo ***que van a recibir el mensaje y colocar unas direcciones que sean de uds. para que puedan comprobar si los mensajes se envían correctamente y para evitar que me lleguen cientos de mensajes de prueba!***

La utilización del operador de asignación ".=" se usa para tomar el valor que ya posee una variable y agregarle más contenido, es decir conserva lo que tiene y le suma otro contenido adicional.

INCLUYENDO ARCHIVOS

Las construcciones include y require son de las más conocidas en php. Con ellas puedes reutilizar porciones de código (script, o simple html) cuantas veces quieras, siendo uno de sus usos más sencillos y típicos el de incluir cabeceras y pies de páginas en un sistema de plantillas.

Include ()

La sentencia include() inserta y evalúa el archivo especificado. Se puede incluir aquí no solamente un archivo de nuestro servidor, sino también una página web remota (indicando la url).

Su uso típico sería:

```
<?php
include("header.php");
?>
```

Include(); llama al archivo header.php y lo inserta en el propio punto del script donde hacemos la llamada.

Tanto si insertamos un archivo con include() o require(), debemos tener en cuenta que PHP pasa a *modo html* hasta el final del mismo, por lo que si el archivo a insertar contiene código php que deba ser evaluado (ejecutado), debe ser encerrado dentro de etiquetas de comienzo y fin de PHP.

Podemos también utilizar varios include anidados (es decir, utilizar include para llamar a otro archivo, dentro del archivo a incluir), con la única precaución de tener en cuenta que los archivos que se van insertando se ejecutan en el entorno del archivo primero que contiene la llamada.

Ejemplo:

Vamos a usar tres archivos, que fusionaremos. Luego observaremos el código de salida.

Archivo 1 : header.php :

```
<html>
<head>
<title> Muestra de includes </title>
</head> <body>
```

Archivo 2: footer.php :

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

Archivo 3: union.php :

```
<?php include("header.php"); ?>
<p>
Hola, este es el contenido.
</p>
<?php include("footer.php"); ?>
```

Ejecutamos unión.php y el resultado sería la suma de los 3 archivos, es decir:

```
<html>
<head>
<title> Muestra de includes </title>
</head>
<body>
<p>
Hola, este es el contenido.
</p>
</body>
</html>
```

FUNCIONES DE LA EXTENSIÓN MYSQLI

La extensión mysqli cuenta con una interfaz dual.

Es compatible con el paradigma de programación procedimental y orientada a objetos.

Los usuarios que migran desde la extensión mysql prefieren la interfaz de procedimiento. La interfaz de procedimiento es similar a la de la extensión mysql. En muchos casos, los nombres de función se diferencian sólo por prefijo. Algunas funciones mysqli toman un identificador de conexión como su primer argumento, mientras que emparejan funciones en la antigua interfaz mysql lo toman como un último argumento opcional.

Veamos una comparación sencilla de la extensión Mysql y la extensión Mysqli:

```
<?php
$mysqli = mysqli_connect("loclahost", "user", "password", "database");
$res = mysqli_query($mysqli, "SELECT * FROM table");
$row = mysqli_fetch_assoc($res);
echo $row['_msg'];

$mysql = mysql_connect("localhost", "user", "password");
mysql_select_db("test");
$res = mysql_query("SELECT * FROM table", $mysql);
$row = mysql_fetch_assoc($res);
echo $row['_msg'];
?>
```

Como se puede ver la migración es muy sencilla y las funciones son prácticamente las mismas, la única diferencia es que en la extensión Mysql, la conexión (en el ejemplo llamada $mysql) es un parámetro optativo que se le puede pasar a la función mysql_query() como segundo parámetro, mientras que en la extensión Mysqli es un parámetro obligatorio y se le debe pasar a la función msqli_query() en primer lugar y como segundo parámetro, el query propiamente dicho.

Pueden encontrar una lista completa de todas las funciones de la extensión aquí: http://php.net/manual/en/mysqli.summary.php

A continuación describiremos algunas de las funciones principales:

Mysqli_connect()

Establece la conexión con el servidor y selecciona la base de datos con la que vamos a trabajar.

```
<?php
```

```
//conection:
$link = mysqli_connect("myhost","myuser","mypassw","mybd") or
die("Error " . mysqli_error($link));
?>
```

Mysqli_query()

Ejecuta el query pasado como parámetro.

```
<?php
//conection:
$link = mysqli_connect("myhost","myuser","mypassw","mybd") or
die("Error " . mysqli_error($link));
//consulta:
$result = mysqli_query($link, "SELECT * FROM table") or die("Error " .
mysqli_error($link));
?>
```

Mysqli_fetch_array()

Lee los resultados como un array asociativo, un array indexado o ambos.

```
<?php
//conection:
$link = mysqli_connect("myhost","myuser","mypassw","mybd") or
die("Error " . mysqli_error($link));
//consulta:
$result = mysqli_query($link, "SELECT * FROM table") or die("Error " .
mysqli_error($link));
//lectura:
while ($row = mysqli_fetch_array($result)){
Var_dump($row);
};
?>
```

Mysqli_fetch_assoc()

Lee los resultados como un array asociativo.

```
<?php
//conection:
$link = mysqli_connect("myhost","myuser","mypassw","mybd") or
die("Error " . mysqli_error($link));
//consulta:
$result = mysqli_query($link, "SELECT * FROM table") or die("Error " .
mysqli_error($link));
//lectura:
while ($row = mysqli_fetch_assoc($result)){
Var_dump($row);
};
```

```
?>
```

Mysqli_fetch_row()

Lee los resultados como un array indexado.

```
<?php
//conection:
$link = mysqli_connect("myhost","myuser","mypassw","mybd") or
die("Error " . mysqli_error($link));
//consulta:
$result = mysqli_query($link, "SELECT * FROM table") or die("Error " .
mysqli_error($link));
//lectura:
while ($row = mysqli_fetch_row($result)){
Var_dump($row);
};
?>
```

Mysqli_num_rows()

Devuelve la cantidad de filas devueltas por un mysql_query int mysqli_num_rows (mysqli_result $result)

```
<?php
//conection:
$link = mysqli_connect("myhost","myuser","mypassw","mybd") or
die("Error " . mysqli_error($link));
//consulta:
$result = mysqli_query($link, "SELECT * FROM table") or die("Error " .
mysqli_error($link));
//lectura:
if (mysqli_num_rows($result)){
while ($row = mysqli_fetch_row($result)){
Var_dump($row);
};
}else{
Echo “No se encontraron datos”;
}
?>
```

Mysqli_error()

Contiene el mensaje de error de la última consulta ejecutada con mysqli_query. Su uso se ve en los ejemplos anteriores:

```
<?php
//conection:
```

```
$link = mysqli_connect("myhost","myuser","mypassw","mybd") or
die("Error " . mysqli_error($link));
//consulta:
$result = mysqli_query($link, "SELECT * FROM table") or die("Error " .
mysqli_error($link));
//lectura:
if (mysqli_num_rows($result)){
while ($row = mysqli_fetch_row($result)){
Var_dump($row);
};
}else{
Echo “No se encontraron datos”;
}
?>
```

BIBLIOGRAFÍA UTILIZADA Y SUGERIDA

- Javier Eguíluz. Introducción a CSS. 1ª Ed. Creative Commons; 2006
- Damián De Luca. HTML5, Entienda el cambio y aproveche su potencial. 1ª Ed. Editorial Fox Andina; 2011
- Alonso Alvarez García. HTML 5, Guía Práctica. 1ª Ed. Anaya Multimedia, Madrid; 2010.

Lo que vimos

En la presente unidad desarrollamos los conceptos necesarios en PHP para trabajar con variables compuestas.
También trabajamos con algunas funciones de PHP, particularmente con las posibles conexiones con la base de datos.

Lo que viene:

En la próxima unidad vamos a trabajar con Javascript y JQuery