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

Centro de e-Learning

EXPERTO UNIVERISTARIO EN MySQL Y PHP NIVEL INICIAL

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

Módulo 2

CONTINUAMOS CON MySQL Y PHP

Funciones (de vectores, definidas por el usuario y propias del lenguaje.)

Presentación de la Unidad:

Aprenderemos a crear nuestras propias funciones, como es el paso de parámetros a una función y como hacemos para definir un valor por defecto para un parámetro.

Vemos algunas funciones importantes y de gran utilidad que nos brinda PHP como por ejemplo la función para hacer posible el envío de mails.

Vemos algunas funciones importantes de PHP para el acceso y administración de datos de las bases de datos.

Objetivos:

- Aprender a definir y utilizar funciones en PHP.
- Aprender el uso de la función mail de PHP que nos hace posible el envío de mails.
- Aprender a utilizar las funciones que nos permitirán establecer una conexión con la base de datos y administrar los datos de la misma.

Temario:

Funciones en PHP
- Funciones y procedimientos
- Funciones declaradas por el usuario
- Parámetros de las funciones
- Parámetros por defecto

Funciones del lenguaje
- Función mail()
- Parámetros opcionales del envío de correo

Incluyendo archivos
- Include
- Require
- Include_once y Require_once

Funciones en PHP para MySQL

Funciones de la extensión Mysqli

Funciones de la extensión Mysql
- 1.- mysql_connect()
- 2.- mysql_select_db()
- 3.- mysql_query()
- 4.- mysql_free_result()
- 5.- mysql_close()
- 6.- mysql_create_db()
- 7.- mysql_fetch_row()
- 8.- mysql_fetch_assoc()
- 9.- mysql_fetch_array()

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 declaradas por el usuario

Dijimos entonces, que las **funciones** son un conjunto de sentencias o instrucciones, que nos permiten pasarles variables (o **parámetros**) y recibir un resultado de vuelta.

Todas las funciones se definen con la palabra ***function*** delante del nombre de la función, luego paréntesis (), que pueden o no contener parámetros dentro, y por último las instrucciones de la función que van entre llaves {}

Pongamos un ejemplo:

```
<?php
function mifuncion(){
instrucciones;
}
?>
```

De esta forma, cada vez que yo llame a mifuncion() esta procesará las instrucciones que haya indicado dentro y me devolverá un resultado.

Las funciones en general son usadas para resumir procesos que son utilizados muchas veces en nuestros scripts, por lo que es conveniente tenerlos resueltos una sola vez en una
función y luego simplemente llamar a dicha función.

Hay dos cosas importantes que debemos saber sobre las funciones, para pasarle datos a una función, esta función debe aceptarlos entre los paréntesis, y para que una función nos devuelva un resultado debemos usar la sentencia return.

Pongamos un ejemplo de una función que acepte dos parámetros, los multiplique entre si y nos devuelva el resultado:

```
<?php
function producto($num1, $num2){
$producto = $num1 * $num2;
return $producto; }
?>
```

$num1 y $num2 son variables internas de la función que tomarán el valor que pasemos al llamar la función, así al llamar la función con 2 y 3 nos devolverá 6 y con 5 y 4, 20.

Veamos el código:

```
$variable1 = producto(10,3); // $variable1 valdrá 30 (10*3)
$variable2 = producto(17,3); // $variable2 valdrá 51
```

Parámetros de las funciones

La información se suministra a las funciones mediante la lista de parámetros, una lista de variables y/o constantes separadas por comas.

PHP soporta el paso de parámetros por valor (el comportamiento por defecto), por referencia, y parámetros por defecto.

Pasar parámetros por valor o por referencia

Por defecto, los parámetros de una función se pasan **por valor**, de manera que, al cambiar el valor de un parámetro dentro de la función, **no** se ve modificado fuera de ella. Para permitir que dichos cambios se vean reflejados fuera de la función, hay que pasar los parámetros **por referencia**.

Para conseguir que un parámetro de una función **siempre** se pase por referencia, hay que anteponer un **ampersand** (**&**) al nombre del parámetro **en la definición** de la función:

```
<?php
function agregar(&$string) { /* Paso por referencia del parámetro.
Los cambios también se verán reflejados fuera de la función*/
$string .= ' y algo más.';
}

$str = 'Esto es una cadena, ';
agregar($str);
echo $str; // Escribe 'Esto es una cadena, y algo más.'
?>
```

En el ejemplofuncion5.php tienen la prueba de ambos casos, pasando parámetros por valor utilizando la función agregar y pasando parámetros por referencia utilizando la función agregarporreferencia:

```
<?php
        function agregar($string) {// Paso por valor del parámetro.
                $string .= ' y algo más.';
        }
        $str = 'Esto es una cadena, ';
        agregar($str);
        echo $str."<br/>"; // Escribe 'Esto es una cadena, '

        function agregarporreferencia(&$str) {// Paso por referencia del
        parámetro.
                $str .= ' y algo más.';
        }

        agregarporreferencia($str);
        echo($str);// Escribe 'Esto es una cadena, y algo más.'
?>
```

Parámetros por defecto

Una función puede definir valores por defecto para los parámetros escalares. Estos valores serán asignados a los parámetros de la función en caso de que el número de parámetros en la llamada a la función sea inferior al número de parámetros en la definición de la función.

```
<?php
function cafe($tipo="cappucino") { /*El valor por defecto del parámetro $tipo
es cappucino*/
return "Haciendo una taza de $tipo.<br>";
}
echo cafe();
/*Llamada a la función sin parámetro. El parámetro tomará su valor por
defecto: cappucino*/
echo cafe("espresso"); //El parámetro tomará el valor espresso
?>
```

El código anterior produce la siguiente salida:
Haciendo una taza de cappucino.
Haciendo una taza de espresso.

Importante: Cuando se usan parámetros por defecto, estos tienen que estar a la derecha de cualquier parámetro sin valor por

defecto; de otra manera las cosas no funcionarán de la forma esperada.

```
<?php
function cafe($temp,$tipo="con leche") {
return "Haciendo café $tipo $temp.<br>";
}
echo cafe("caliente"); //Escribe: 'Haciendo café con leche caliente'
?>
```

Devolver valores

Para que una función devuelva un valor se emplea la instrucción opcional return. Puede devolverse cualquier tipo de valor, incluyendo listas y objetos.

```
<?php
function cuadrado($num) {
return $num * $num; // Devuelve el cuadrado de $num
}
echo cuadrado (5); //Escribe: 25
?>
```

No es posible devolver múltiples valores desde una función, pero un efecto similar se puede conseguir devolviendo una lista, para ello se emplea la función de PHP list(), veamos un ejemplo:

```
<?php
function numeros() {
return array (0,1,2,3);
}
list ($cero, $uno, dos, $tres) = numeros();
echo ($cero.“,“);
echo ($uno.“,“);
echo ($dos.“,“);
echo ($tres);
?>
```

Funciones del lenguaje

Desarrollaremos a continuación algunas de las funciones más utilizadas, otras las iremos viendo a lo largo de los siguientes módulos.

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("veropineyro@hotmail.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 veropineyro@hotmail.com,
con copia a veronicapineyro@gmail.com y con copia oculta para

vpineyro@yahoo.com.ar y veronica@ideasenpixel.com.ar

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

```
<?php
$destinatario = "veropineyro@hotmail.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: Veronica Piñeyro < veropineyro@hotmail.com >\r\n";
//dirección de respuesta, si queremos que sea distinta que la del remitente
$headers .= "Reply-To: veronica@gmail.com \r\n";
//direcciones que recibián copia
$headers .= "Cc: veronicapineyro@gmail.com \r\n";
//direcciones que recibirán copia oculta
$headers .= "Bcc: vpineyro@yahoo.com.ar, veronica@ideasenpixel.com.ar
\r\n";
mail($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>
```

Require

A partir de la versión de PHP 4.0.2 y posteriores, ambas construcciones se comportan exactamente de la misma manera, con la única diferencia de que si el archivo llamado no existe, include solo da una advertencia, y sigue ejecutando el código, mientras que require produce un error fatal e interrumpe la ejecución.

Ejemplo:

Include:

```
<?php include("noexiste.php");
echo ("Hola. El script siguió!");
?>
```

Y lo que obtendremos:

```
Warning: include(noexiste.php) [function.include]: failed to open stream: No such file or directory in probando.php on line 2
```

```
Warning: include() [function.include]: Failed opening 'noexiste.php' for
inclusion (include_path='.;C:\php5\pear') in probando.php on line 2
Hola. El script siguió!
```

El script continuó más allá del error.

Require:

```
<?php require("noexiste.php");
echo ("Hola. El script siguió!");
?>
```

Y aquí obtenemos:

```
Warning: require(noexiste.php) [function.require]: failed to
open stream: No such file or directory in probando.php on line
2
Fatal error: require() [function.require]:
Failed opening required 'noexiste.php'
(include_path='.;C:\php5\pear') in probando.php on line 2
```

Vemos que con require no se ejecutó la parte del script posterior al error.

NOTA: En php 3 y anteriores, las funciones include y require se diferenciaban por un asunto aún mayor: Include podía ser usado condicionalmente,
mientras que require se ejecutaba "a la fuerza", esto ya no es así.

Include_once y Require_once

Estas construcciones presentan como única diferencia que la inclusión del archivo se ejecuta una sola vez (aunque a lo largo de la ejecución del script existan otras llamadas al mismo), lo que es útil para evitar conflictos con redefiniciones de funciones o nombres de variables.

Nota: Los archivos a incluir no tienen que ser necesariamente archivos PHP. Pueden ser de cualquier tipo.

Funciones en PHP para MySQL

Existen en PHP varias extensiones para acceder a las tablas mysql, la mas conocida de todas es la extensión mysql. Sin embargo esta extensión ha sido declarada como obsoleta a partir de PHP 5.5 y se recomienda no escribir código nuevo con ella.

En el manual de PHP se recomienda reemplazar esta extensión por mysqli o por PDO.

Mysqli posee dos versiones, una procedimental y una orientada a objetos. PDO es solo orientada a objetos.

A continuación veremos las funciones de la extensión mysqli en su versión por procedimientos.

Funciones de la extensión Mysqli

A continuación mencionamos esta comparación entre Mysql y Mysqli para aquellos que vienen acostumbrados a trabajar con las funciones de la vieja extensión o que han visto y probado algun ejemplo con la misma. Aunque encuentren scripts en la web que todavía utilizan la extensión mysql les conviene no utilizarla. En el curso utilizaremos siempre la extensión Mysqli. Incluso aclaramos que las nuevas versiones de PHP arrojan un Warning “Deprecated…” indicando que la extensión está obsoleta.

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 el 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 extensiónMysqli

```php
<?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";
}
?>
```