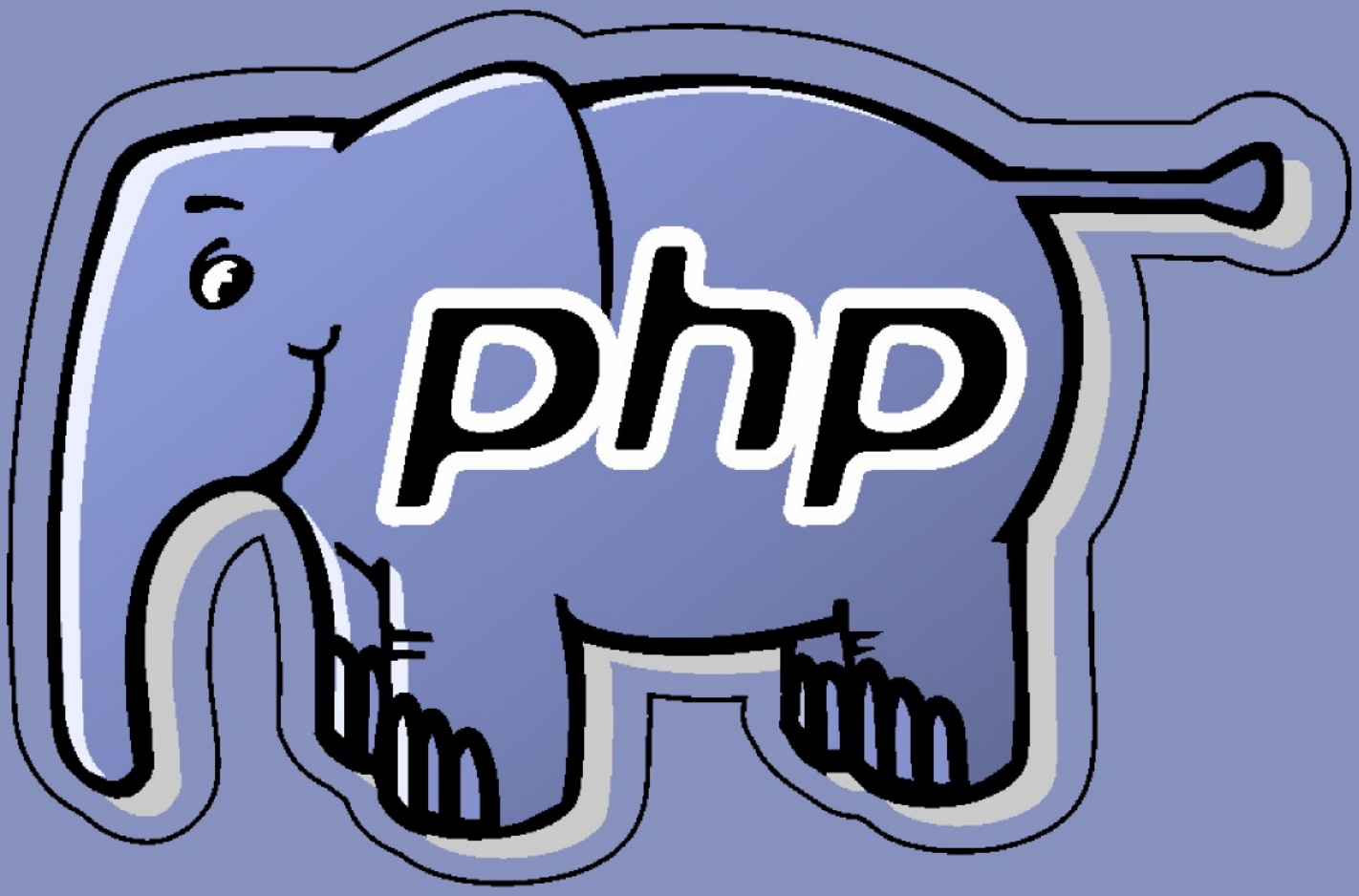
Programación en



Versión 2.0

php

Gugler
Laboratorio de Investigación Gugler

FCyT
Facultad de Ciencia y Tecnología

UADER
Universidad Autónoma de Entre Ríos

CONTENIDO

Capítulo 2

Programación Web

Uploads de Archivos

Definimos *upload de archivo* al proceso mediante el cual podemos adjuntar cualquier tipo de archivo dentro de un formulario para poder almacenarlo posteriormente en el servidor donde se encuentra alojada la aplicación web. La idea detrás de ésto es que los usuarios puedan incluir archivos completos a mediante el uso de formularios. Los archivos pueden ser archivos de texto, archivos de imagen u otros datos.

Escribir un formulario HTML que permita el envío de archivos es bastante simple. Para ello haremos uso de la etiqueta *input* de tipo *file* de la siguiente forma:

```
<input type=”file” name='fotografia >
```

Esta etiqueta nos despliega una ventana de búsqueda para que podamos localizar y adjuntar el archivo que queramos enviar. Ejemplo:

```
<html>
    <head>
        <title>Ejemplo de form con INTUP TYPE</title>
    </head>
    <body>
        <form action="subirArchivo.php" enctype=”multipart/form-data”
        method="POST" >
            Seleccione una Foto:
            <input type="file" name="foto" /><br />
            <input type="submit" value="Enviar Foto" />
        </form>
    </body>
</html>
```

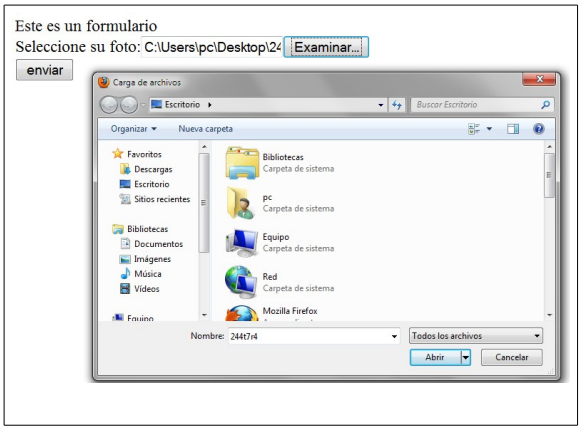


Figura1: elemento FILE

Deberemos de especificar el atributo *enctype="multipart/form-data"* dentro del formulario para habilitar el envío de archivos, como se ve en el ejemplo:

```
<form action="subirArchivo.php" enctype="multipart/form-data"  method="POST" >
        Seleccione una Foto:
                <input type="file" name="foto" /><br />
                <input type="submit" value="Enviar Foto" />
</form>
```

Por defecto un formulario utiliza un tipo de contenido enctype=*"application/x-www-form-urlencoded"* el cual no es eficiente para enviar grandes cantidades de datos binarios o textos que contenga caracteres no ASCII. Para enviar formularios que contengan ficheros, datos no ASCII y datos binarios utilizaremos el tipo de contenido enctype=*"multipart/form-data"*.

Del lado del servidor deberemos de programar nuestro script *subirArchivo.php* para poder manipular el mismo.

Procesando archivos con PHP

Estudiaremos en esta sección de que manera y mediante el uso de que estructuras podremos manipular los archivos recibidos en nuestro script PHP.

$_FILES

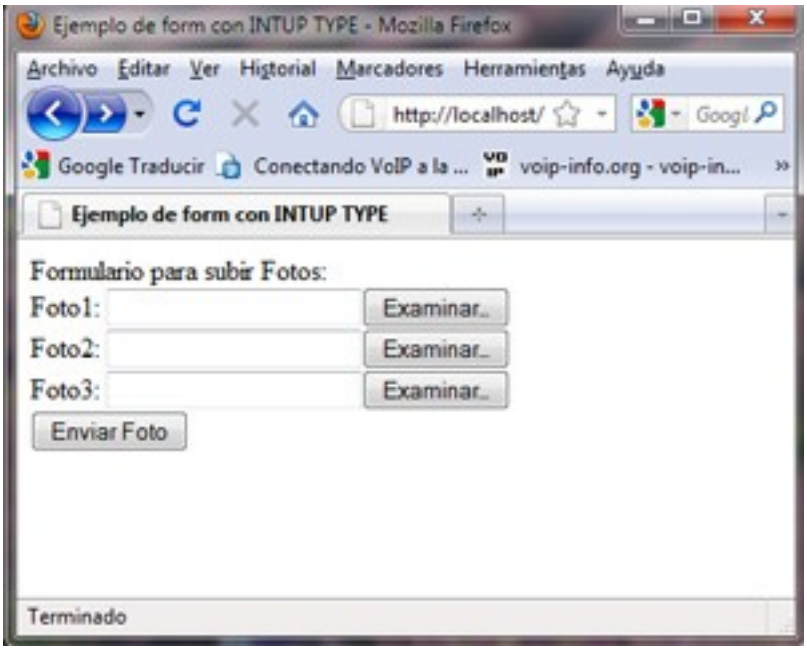
Cuando se envían uno o varios archivos por medio de un formulario a un script PHP, se

crea un arreglo llamado $_FILES[] el cuál contendrá, por cada archivo enviado, un arreglo con distinta información del mismo. Ejemplo:

Dado el siguiente formulario:

```
<html>
    <head>
        <title>Ejemplo de form con INTUP TYPE</title>
    </head>
    <body>
        <form action="subirArchivo.php" enctype="multipart/form-data"
        method="POST">
            Formulario para subir Fotos:<br />
                Foto1: <input type="file" name="foto1" /><br />
                Foto2: <input type="file" name="foto2" /><br />
                Foto3: <input type="file" name="foto3" /><br />
            <input type="submit" value="Enviar Foto" name="bt_submit" />
        </form>
    </body>
</html>
```

El cual mostraría algo así:



En PHP quedará definido un arreglo llamado $_FILES el cual contendrá la siguiente estructura:

```
Array (
    [foto1] => Array (
        [name] => foto1.jpg
```

```
                [type] => image/jpeg
                [tmp_name] => C:\xampp\tmp\php2277.tmp
                [error] => 0
                [size] => 172905 )
        [foto2] => Array (
                [name]=>foto2.JPG
                [type] => image/jpeg
                [tmp_name]=>C:\xampp\tmp\php2287.tmp
                [error] => 0
                [size]=>2759123 )
        [foto3] => Array (
                [name]=>foto3.jpg
                [type]=>image/jpeg
                [tmp_name]=>C:\xampp\tmp\php22D6.tmp
                [error]=>0
                [size] => 172905 )
)
```

Como podemos ver, el arreglo posee tres elementos distintos (foto1, foto2 y foto3). Cada uno de estos elementos corresponde unívocamente a los *input file* definidos en el formulario. Además podemos ver que cada elemento del array es otro array con información del archivo subido. Tomemos por ejemplo foto1:

```
[foto1] => Array (
        [name] => foto1.jpg                          ← nombre del archivo.
        [type] => image/jpeg                         ← mime type del archivo.
        [tmp_name] => C:\xampp\tmp\php2277.tmp       ← nombre temporal del archivo.
        [error] => 0                                 ← mensaje de error.
        [size] => 172905                             ← tamaño en bytes.
)
```

Mover o copiar el archivo

Cuando hacemos un upload de archivo éste es copiado en un directorio temporal dentro del servidor web. Este directorio es definido en el archivo de configuración de PHP (php.ini) mediante el atributo *upload_tmp_dir.* **De no especificar lo contrario el archivo subido será eliminado al finalizar la ejecución del script**.

La manera de realizar esto es mediante la función:

move_uploaded_file($nombre_temporal,$directorio).

Por ejemplo:

Definimos nuestro formulario:

```
<html>
	<head>
		<title>Ejemplo de form con INTUP TYPE</title>
	</head>
	<body>
		<form action="subirArchivo.php" enctype="multipart/form-data"
		method="POST">
			Formulario para subir Fotos:<br />
				Foto1: <input type="file" name="foto1" /><br />
			<input type="submit" value="Enviar Foto" name="bt_submit" />
		</form>
	</body>
</html>
```

y en nuestro script subirArchivo.php:

```
<?php
	if(is_uploaded_file($_FILES['foto']['tmp_name'])){
	  $archivosFotos="fotos/";
	  $nombre=$_FILES['foto']['name'];
	    if(move_uploaded_file($_FILES['foto']['tmp_name'],$archivosFotos.$nombre)){
	        print ("El archivo fue movido al directorio ".$upload_dir);
	    }else{
	        print ("No se pudo mover el archivo al directorio ".$upload_dir);
	    }
	}else{
		print ("No se pudo subir el archivo al servidor.");
	}
?>
```

En primer lugar aparece la función *is_upload_file()* la cual indica por medio de *true* o *false* si un archivo ha sido subido mediante HTTP POST. Haciendo uso de esta verificamos que se halla transferido el archivo desde el navegador web hacia el servidor.

Luego definimos un directorio donde mover la imagen subida (recuerde que se sube de manera temporal, si no la copiamos a otro directorio se borra). Dicho directorio lo definimos en la variable *$archivosFotos*. De la misma forma, haciendo uso del array $_FILES extraemos el nombre del archivo. Esto lo hacemos en la variable $nombre.

Una vez que sabemos donde guardar el archivo ($*archivosFotos)* y con que nombre (*$nombre*) procedemos a ejecutar la función *move_uploaded_file.* Evaluando el resultado que arroja la misma (*true* o *false*), mostraremos un mensaje satisfactorio o no según si se pudo mover o no el archivo.

Restringir el tamaño de los archivos

El hecho de permitir que los usuarios de nuestra aplicación suban archivos al servidor es una alternativa muy buena pero no debemos dejar de realizar algunos controles sobre la misma. Uno de estos controles corresponde al tamaño de los archivos a enviar.

La restricción sobre el tamaño de los archivos a enviar al servidor es una operación que se puede realizar tanto en la parte cliente (en el navegador web) como en el servidor (mediante la configuración de PHP).

Del lado del Cliente

En el lado cliente podremos hacer esto agregando a nuestro formulario una etiqueta *INPUT* de tipo *hidden* y nombre *MAX_FILE_SIZE* en cuyo valor (propiedad *value*) podremos especificar el tamaño en bytes que permitiremos.

```
<input type="hidden" name="MAX_FILE_SIZE" value="102400">
```

Si en tamaño del archivo superase el definido en esta etiqueta, el formulario será enviado pero no se adjuntará el archivo.

Del lado Servidor

Del lado servidor podemos hacer uso de la información que nos provee el array $_FILES. Si recordamos un ejemplo:

```
[foto1] => Array (
        [name] => foto1.jpg                          ← nombre del archivo.
        [type] => image/jpeg                         ← mime type del archivo.
        [tmp_name] => C:\xampp\tmp\php2277.tmp       ← nombre temporal del archivo.
        [error] => 0                                 ← mensaje de error.
        [size] => 172905                             ← tamaño en bytes.
)
```

Vemos que uno de los campos del arreglo es el tamaño en bytes (*size).* Con esta información podremos generar una estructura *if – else* que valide el mismo y permita o no mover el archivo a un directorio seguro o simplemente dejarlo para que sea eliminado. Veamos un ejemplo:

Definimos nuestro formulario:

```
<html>
        <head>
                <title>Ejemplo de form con INTUP TYPE</title>
        </head>
```

```
        <body>
            <form action="subirArchivo.php" enctype="multipart/form-data"
            method="POST">
                Formulario para subir Fotos:<br />
                    Foto1: <input type="file" name="foto1" /><br />
                <input type="submit" value="Enviar Foto" name="bt_submit" />
            </form>
        </body>
    </html>
```

y en nuestro script subirArchivo.php:

```
<?php
    if($_FILES['foto']['size']<=10000){
       $archivosFotos="fotos/";
       $nombre=$_FILES['foto']['name'];
          if(move_uploaded_file($_FILES['foto']['tmp_name'],$archivosFotos.$nombre)){
              print ("El archivo fue movido al directorio ".$upload_dir);
          }else{
              print ("No se pudo mover el archivo al directorio ".$upload_dir);
          }
    }else{
            print ("El archivo que intenta subir es muy grande");
    }
?>
```

En el ejemplo mediante *$_FILES['foto']['size']* verificamos que el tamaño sea menor o igual que 10000. Si no se cumple, no se guardará el mismo en el directorio fotos/.

Por defecto en PHP

PHP trae por defecto en su archivo de configuración (php.ini) una linea de donde restringir el tamaño de archivos que PHP permitirá subir. Tenga en cuenta esta configuración para sus aplicaciones ya que querría que las mismas suban archivos muy grandes y si superan el tamaño definido en este archivo, el script le dará error.

El atributo que define el tamaño permitido por PHP para el upload de archivo es el *UPLOAD_MAX_FILESIZE* y su valor por defecto podría variar.

Sesiones

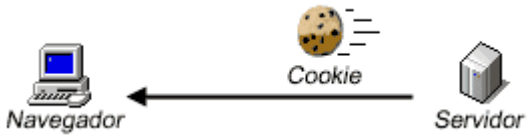
Introducción

HTTP es un protocolo sin estado, lo que significa que el servidor web no sabe si dos solicitudes proviene del mismo usuario. Cada petición es manejada sin tener en cuenta el contexto en que sucede. Las sesiones sirven para crear una medida de estado entre las solicitudes, incluso cuando se producen a intervalos de tiempo grandes unas de otras, lo que nos permitirá realizar un seguimiento sobre la navegación que el usuario está realizando.

Sesiones en una aplicación web

Una sesión en una aplicación web consiste en controlar el acceso y la navegación de cada usuario sobre la misma mediante la asignación de un identificador único para cada uno. Con ello se puede conseguir hacer un seguimiento de cada usuario a lo largo de su visita en nuestra página web, desde que entra hasta que sale.

El soporte de sesiones consiste en la posibilidad de conservar datos a lo largo de toda la navegación que un usuario realiza sobre nuestro sitio. A cada usuario que accede a nuestro sitio se le asigna un identificador único denominado *session id*. Este identificador es almacenado en una cookie en el navegador del cliente y en una variable de sesión del lado del servidor.



Las cookies son pequeños archivos de texto que se guardan en el disco duro del visitante de una página web y que contienen información generada por el servidor. Esta información puede ser luego accedida por el servidor para ser utilizada.

Sesiones en PHP

PHP nos provee de varias funciones para el manejo de sesiones. Además, una vez que creamos una sesión se define de manera automática una variable global de nombre *$_SESSION*. Esta variable será un arreglo en el cual podremos ir guardando los datos que nosotros necesitemos.

La forma de trabajo de las sesiones sigue la siguiente metodología:

si existe una sesión, la utilizo
sino la creo

Crear una sesión

Para crear una sesión haremos uso de la función *session_start()* la cual creará una cookie en el navegador y al mismo tiempo generará una variable dentro del servidor ambas identificadas por un session id. Ejemplo:

```
<?php
        session_start();
        echo “El session id creado es: “;
        echo  session_id();             //Resultado jmjk09rs0rptfk1qsf8kkhdsl1
?>
```

Podemos ver además que mediante la función *session_id()* obtendremos el *id* asignado a dicha sesión. Si quisiésemos asignarle un nombre específico a la misma, haremos uso de la función *session_name()* antes de crearla. Ejemplo:

```
<?php
        session_name('Mi-Session');
        session_start();
        echo “El session id creado es: “;
        echo  session_name();           //Resultado Mi-Session
?>
```

Guardar valores en una sesión

Como explicamos anteriormente, al crear una sesión PHP genera un arreglo de nombre *$_SESSION[]* dentro del cual podremos ir almacenando los valores que nosotros necesitemos. Ejemplo:

```
<?php
        session_start();
                $_SESSION['apellido']=”Colón”;
                $_SESSION['nombre']=”Cristobal”;
        echo $_SESSION['nombre'] $_SESSION['apellido'];
?>
```

De esta manera, quedará en el arreglo *$_SESSION* los dos valores por nosotros ingresados. Si quisiésemos recuperar estos valores en otro script solamente deberíamos invocar *session_start()* y ya estaría a disposición *$_SESSION*,

Veamos a continuación otro ejemplo en el cual definiremos una variable de sesión llamada *contador* la cual se inicializará en 0 la primer vez que se accede al script y se irá incrementando a medida que volvamos a ejecutar el mismo. Ejemplo:

```
<?php
        session_start();
```

```
        if (!isset($_SESSION['contador'])) {
                $_SESSION['contador'] = 0;
        } else {
                $_SESSION['contador']++;
        }
        echo $_SESSION['contador'];
?>
```

Como vemos en el ejemplo, la función *isset()* evalúa si se encuentra definida *$_SESSION['contador']* si no esta lo crea y le asigna el valor 0. La segunda vez que ejecutemos el script (sin cerrar el navegador, de lo contrario se borra la sesión) saltará esta validación incrementando su valor en 1.

Login de usuario

Con los conceptos vistos hasta el momento estamos en condiciones realizar a un pequeño formulario que permita realizar un login por usuario y contraseña. Ejemplo:

```
<?php
        session_start();
        if (!isset($_SESSION['logueado'])){
                $user="cristobal";
                $pass="colon";
                if (($_POST['usuario']!='')&&($_POST['password']!='')){
                        if (($_POST['usuario']==$user)&&($_POST['password']==$pass)){
                                $msj="<font color='green'>Usted se ha logueado!!!</font>";
                        }else{
                                $msj="<font color='red'>usuario o contraseñas invalidos</font>";
                        }
                }else{ $msj='';}
        }else{
                $msj="<font color='green'>ya estás logueado!!!</font>";
        }
?>

<html>
        <head>
                <title>Formulario de Login</title>
        </head>
        <body>
                <form action="loginForm.php"  method="post">
                        <table>
                                <tr><td colspan="2">Formulario de Login:</td></tr>
                                <tr>
                                        <td>Usuario:</td>
                                        <td><input type="text" name="usuario" /></td>
                                </tr>
                                <tr>
                                        <td>Contraseña:</td>
                                        <td><input type="password" name="password" /></td>
                                </tr>
```

```
                    </table>
                    <input type="submit" value="Login" />
                </form>
                <?php echo $msj;?>
        </body>
</html>
```

Si analizamos el script (loginForm.php) vemos que el mismo está compuesto por dos partes: la primer parte sentencias PHP que controlan el login y la sesión del usuario y la segunda el formulario propiamente dicho. Cuando accedamos por primer vez, al no existir una sesión y al no encontrar datos en las variables *$_POST['usuario']* y *$_POST['password']* se mostrará el formulario sin mas nada.

Completando los datos y enviándolos (note que los datos son enviados al mismo script) al encontrar valores en las variables éstas serán comparados con los datos guardados en $user y $pass. De ser correctos los mismos aparecerá un mensaje avisando que se ha logueado bien. En otro caso, mostrará distintos mensajes de error.

Remover variables de una sesión

La función *session_unset()* elimina todas las variables asociadas a una sesión determinada. Esta función no destruirá la función creada, sino todos los valores definidos dentro del array *$_SESSION*. Para poder utilizarla, primero deberemos recuperar los datos de la sesión mediante *session_start()*. Ejemplo:

```
<?php
        session_start();
        $_SESSION['valor']=4;
        echo “el valor de arrelgo SESSION[valor] es: “;
        echo $_SESSION['valor'];
        echo “<br />Ahora destruimos la session”;
        session_unset();
        echo “el valor de arrelgo SESSION[valor] es: “;
        echo $_SESSION['valor'];
?>
```

Destruir una sesión

Para destruir una sesión existe la función *session_destroy().* Esta función destruirá la sesión creada pero no borrará los valores de las variables de sesión creadas. Para esto utilizamos *session_unset()*. Como se puede ver, el hecho de destruir una sesión se debería de realizar haciendo uso de las dos funciones. Ejemplo:

```
<?php
        session_start();                    // creamos la sesión
        $_SESSION['valor']=4;               // creamos una variable de session
        echo “el valor de arrelgo SESSION[valor] es: “;
        echo $_SESSION['valor'];
        echo “<br />Ahora destruimos la session”;
        session_unset();
        session_destroy();
        echo “el valor de arrelgo SESSION[valor] es: “;
        echo $_SESSION['valor'];
?>
```

Inclusión de scripts

Introducción

La inclusión de scripts nos permite insertar el contenido de un script (PHP o HTML) en otro antes de que el servidor los ejecute. Es ideal para incluir cabeceras y pies de página en todas las páginas que componen nuestro sitio y así, si alguna vez se debemos modificar algo, solamente deberemos cambiar un solo archivo. También se suele utilizar para menúes o cualquier parte de código que se repita en todo el sitio.

PHP nos ofrece algunas funciones para realizar este tipo de tareas: *include()*, *require()*, *include_once()* y *require_once()*.

include()

La función *include()* toma todo el contenido de un archivo especificado y lo inserta dentro del cuál es invocado. Si ocurriese un error (por ejemplo que se especifique mal la ruta del archivo) emite un mensaje de warning y continúa con la normal ejecución del script. Ejemplo:

Definamos header.php

```
<?php
        echo “Este es el encabezado de la Página. “;
        echo “Podemos incluir esta sección en todas las demás páginas. ”
        echo “La Fecha de hoy : “.date('d-m-Y')
?>
```

Nuestro index.php quedaría así:

```
<html>
    <head>
        <title>Uso de Include y Required</title>
    </head>
    <body>
        <table align=”center” width=”100%”>
            <tr>
                <td><?php include('header.php'); ?></td>
            </tr>
            <tr>
                <td>CUERPO DE LA PÁGINA</td>
            </tr>
    </body>
</html>
```

require()

Básicamente la función require*()* realiza la misma tarea que la función *include(),* toma todo el contenido de un archivo especificado y lo inserta dentro del cuál es invocado.

La diferencia radica en que si ocurre genera un *fatal error* y detiene la ejecución del script. Ejemplo:

Definamos header.php

```
<?php
    echo “Este es el encabezado de la Página. “;
    echo “Podemos incluir esta sección en todas las demás páginas. ”
    echo “La Fecha de hoy : “.date('d-m-Y')
?>
```

Nuestro index.php quedaría así:

```
<html>
    <head>
```

```
        <title>Uso de Include y Required</title>
    </head>
    <body>
        <table align=”center” width=”100%”>
            <tr>
                <td><?php require('header.php'); ?></td>
            </tr>
            <tr>
                <td>CUERPO DE LA PÁGINA</td>
            </tr>
    </body>
</html>
```

include_once() y require_once()

Realizan la misma funciones que include() y require() pero solamente incluirán una única vez el código. Es decir, si tuviésemos mas de una llamada a dichas funciones, solamente la primer vez se incluirían.