

TEMA 03. FORMULARIOS Y TRATAMIENTO DE FICHEROS EN PHP. Parte 1

Desarrollo web entorno servidor

CFGS DAW

Autor: Cristina Alvarez

Revisado por: Vanessa Tarí Costa – vanessa.tari@ceedcv.es

2020/2021

Versión:221021.1953

Licencia

Reconocimiento – NoComercial – CompartirIgual (by-nc-sa):

No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

☐ Importante

☐ Atención

☐ Interesante

Reconocimiento.

Reconocimiento a mi compañera del CEED Cristina Alvarez por sus apuntes del curso 2016-17.

Revisiones

- 31/10/2019. Apartado 2.5. Pasar una array de valores por get.
- 24/10/2020. Areglados ejemplos de formularios

Contenido

Contenido	3
1. FORMULARIOS HTML	4
1.1 Declaración	4
1.2 Método get	5
1.3 Método post	5
1.4 Pasar una array de valores por get	6
1.5 Campos de entrada de datos	7
1.6 Botones	8
2. ENVÍO Y RECEPCIÓN DE FORMULARIOS	11
2.1 Simple: con varias páginas web	11
2.2 Redirección de páginas web	13
3. CONTROLES	16
3.1 Botón enviar (submit button)	16
3.2 Entradas de datos (input / textarea)	17
3.3 Casilla de verificación (checkbox)	17
3.4 Botón radio	17
3.5 Menú	18
4. CARGAR ARCHIVOS	19
5. RECOGIDA DE DATOS	23
6. SEGURIDAD EN LAS ENTRADAS	26
7. COMPROBACIÓN DE DATOS CON FUNCIONES	34
8. COMPROBACIÓN DE DATOS CON EXPRESIONES REGULARES	38
9. BIBLIOGRAFÍA	45

UD03. Formularios. Parte 1

1. FORMULARIOS HTML

1.1 Declaración

Todo formulario se encaja en la siguiente sintaxis:

```
<form name=" " method= " " action = " ">  
//contenido del formulario  
</form>
```

Así, se destacan los siguientes elementos:

- Etiquetas de inicio/fin: en ellas se pueden ubicar como mínimo tres elementos:
 - a) **Name**: el nombre del formulario. No es obligatorio pero sí recomendable.
 - b) **Method**: el método que se emplea para enviar los datos (GET/POST)
 - c) **Action**: la página destino que procesará los datos que envía el formulario. Suele ser un fichero de extensión php. También se puede dejar en blanco.
- Campos de entrada de datos
- Botones: para realizar las acciones

Un ejemplo sería:

```
<form name="formulario" method="get" action="procesar.php"> Campos de  
entrada de datos  
Botones de acciones  
</form>
```

Para que el servidor y cliente se entiendan deben entender unos mismos protocolos de comunicación. GET y POST son métodos del protocolo http para el intercambio de información entre ambos agentes. Seguidamente vamos a ver los métodos get/post.

☐ Interesante. Diferencias entre get y post

<https://difiere.com/la-diferencia-get-post/>

1.2 Método get

El método GET es el más utilizado en la actualidad. Lo que hace es pedir al servidor web que le devuelva al cliente la información identificada en la petición URL (un documento HTML, una imagen, una consulta a una base de datos...). El servidor procesa la petición y devuelve al cliente el resultado.

Sus características principales son:

- Es un método de invocación (recupera información)
- Todos los datos de la petición son **visibles** en la barra de direcciones del navegador (van en la cadena de solicitud query string). Cuando se envía el formulario se carga en el navegador la dirección especificada URL como action, se le añade un ? y se incluyen los datos del formulario concatenados con &. Un ejemplo sería:

```
http://site/procesa.php?name1=value1&name2=value2&name3=value3
```

- Los valores se guardan en un array asociativo llamado **\$_GET**.
- Solo se aceptan caracteres ASCII, los especiales (ej. Ñ, ñ, é,<,&...) se deben enviar con el formato %código_hexadecimal_del_carácter.
- Los datos a transferir se envían en una cadena que como máximo puede tener 2.083 caracteres.

Esto hace que GET sea usado cuando los datos no producen modificaciones en el servidor tal como ocurre en consultas en bases de datos, búsquedas y similares.

1.3 Método post

Por su parte, el método POST es usado para enviar información a un servidor web. Ejemplos de lo que se conoce como “posting” son el envío de datos para formularios de autenticación, entradas de datos o especificación de parámetros para algún software de servidor. Lo que haga dependerá de la petición URL. Sus principales características son:

- Es un método de envío de información
- No tiene limitaciones de tamaño en los campos de caracteres: pueden ser no ASCII (permite imágenes, archivos...)
- Los datos del formulario **no se visualizan en la barra de direcciones** del navegador: se envían en el cuerpo de mensaje.
- Los valores se guardan en el array asociativo **\$_POST**.
- Como en GET; los caracteres especiales hay que traducirlos a ASCII.
- Es necesario indicar el tipo de codificación en el <form> con el atributo enctype, que puede tener dos valores:
 - application/x-www-form-urlencoded: no permite enviar archivos. Es el valor por defecto.
 - multipart/form-data: **sí permite enviar archivos.**

En resumen, la principal diferencia entre ambos métodos es cómo codifican la información. GET envía las variables dentro de la URL de la página, mientras que POST las codifica en el cuerpo de la petición HTTP (viajan ocultas). Por ello, se usa POST como método de transferencia de datos es el más habitual cuando se utilizan formularios o cuando los datos transferidos producen algún tipo de modificación en los dos datos almacenados en el servidor tales como altas en bases de datos, modificaciones, etcétera.

☐ Interesante

PHP tiene un array global que es la unión de GET y POST. Veamos un resumen:

- * \$_GET: parámetros enviados mediante GET o en la URL
- * \$_POST: parámetros enviados mediante POST
- * \$_REQUEST: unión de \$_GET y \$_POST y \$_COOKIES

1.4 Pasar una array de valores por get

Podemos pasar como parámetros valores por get serializando el vector. Para ello usaremos la función `serialize($vector)` para crear la url y `unserialize($vector)` para extraer su contenido. Seguidamente se muestra un ejemplo:

vectorSerializado1.php

```
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<?php
$miArray = array("uno", "dos");
echo "<a href='vectorSerializado2.php?miArray=" . serialize($miArray) .
"'>ir a mipagina.php pasando como parametro un array</a>";
?>
</body>
</html>
```

vectorSerializado2.php

```
<html>
```

```

<head>
<meta charset="UTF-8">
<title></title>
</head>
<body>
<?php
$miArray = unserialize($_GET["miArray"]); echo "<pre>";
print_r($miArray); echo "</pre>";
?>
</body>
</html>

```

1.5 Campos de entrada de datos

Una vez declarado el formulario, debemos poner un campo para cada uno de los datos que queramos recibir. Hay de seis tipos:

TIPO	DESCRIPCIÓN	EJEMPLO
Texto	Recibe una línea de texto	<code><input type="text" name="nombre"></code>
Área de texto	Permite recibir textos largos de varias líneas	<code><textarea name="opinion"></textarea></code>
Selección única	Permite elegir una opción entre varias. Si se quieren excluyentes deben tener el mismo nombre pero diferentes valores	<code><input type="radio" name="sexo" value="hombre"></code> <code><input type="radio" name="sexo" value="mujer"></code>
Selección multiple	Permite elegir una o más opciones de entre varias de un cuadro	<code><input type="checkbox" name="cine"></code> <code><input type="checkbox" name="musica"></code> <code><input type="checkbox" name="lectura"></code>
Lista de selección	Permite elegir una o más opciones de entre varias de una lista desplegable	<code><select name="sexo"></code> <code><option>hombre</option></code> <code><option>mujer</option></code> <code></select></code>
Campo oculto	Permite pasar información entre páginas PHP independientes	<code><input type="hidden" name="referencia"></code>

En todos

ellos es necesario usar el atributo name. Éste guarda el nombre de la variable con el que vamos a recibir los datos. El atributo value es opcional e indica el valor predefinido que se les da.

1.6 Botones

Todos los formularios han de tener al menos un **botón de submit** con el que enviar los datos una vez han sido rellenados. Éste se implementa tal y como se muestra:

```
<input type="submit" name="Enviar">
```

Por otro lado, se suele añadir un **botón de reset**. Al pulsarlo, se devuelven los campos a su estado y/o valor inicial:

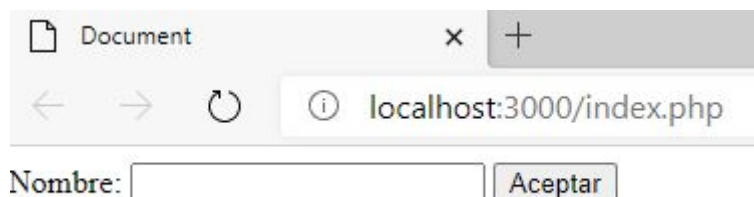
```
<input type="reset" name="Borrar todo">
```

Hagamos nuestro primer ejemplo. Probemos a crear un formulario muy simple. Escribe el siguiente código y guárdalo con el nombre “prueba2.html”

Ejemplo1.

```
<html>
<head>
  <title>prueba2.html</title>
  <meta charset="UTF-8">
</head>
<body>
  <form action="response.php" method="post">
    Nombre: <input type="text" name="Nombre">
    <input type="submit" value="Aceptar">
  </form>
</body>
</html>
```

Daríá lugar a:



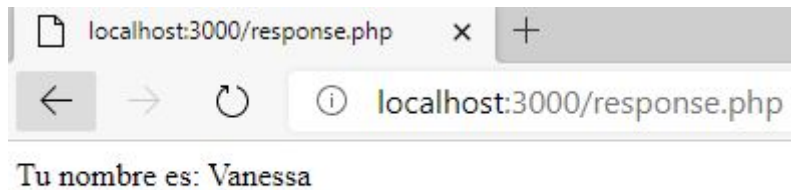
The screenshot shows a web browser window with a single tab titled 'Document'. The address bar displays 'localhost:3000/index.php'. Below the address bar, the rendered HTML form is visible. It consists of the label 'Nombre:' followed by a text input field and an 'Aceptar' button.

Ahora

escribiremos el fichero “response.php”

```
<?php
$tuNombre= $_REQUEST['Nombre'];
echo "Tu nombre es: $tuNombre";
```

Y al escribir nuestro nombre y pulsar el botón Aceptar veríamos en pantalla:



Ejemplo 2:

Ahora vamos a probarlo con un ejemplo que iremos desarrollando. Para ello, creamos un formulario que vamos a guardar con el nombre **encuesta.html** y que usaremos a lo largo de este tema:

```
<!DOCTYPE html>
<html lang="es">

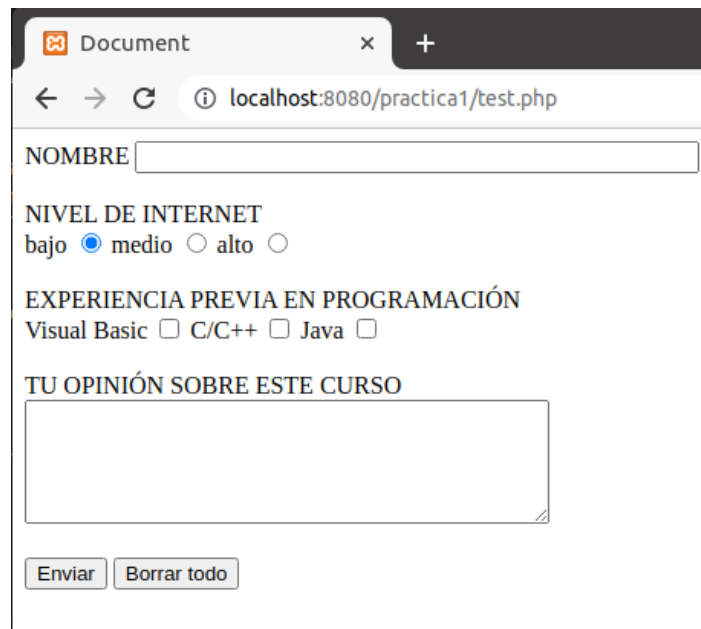
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <form name="encuesta" method="post" action="verificar.php">
    NOMBRE <input type="text" name="nombre" size="43"><br>
    <br>
    NIVEL DE INTERNET<br>
    bajo <input type="radio" name="nivel" value="bajo" checked>
    medio <input type="radio" name="nivel" value="medio">
    alto <input type="radio" name="nivel" value="alto"><br>
    <br>
    EXPERIENCIA PREVIA EN PROGRAMACIÓN<br>
    Visual Basic <input type="checkbox" name="basic">
    C/C++ <input type="checkbox" name="c_cplus">
    Java <input type="checkbox" name="java"><br>
    <br>
    TU OPINIÓN SOBRE ESTE CURSO<br>
    <textarea name="opinion" cols="40" rows="5"></textarea><br>
    <br>
    <input type="submit" value="Enviar">
    <input type="reset" value="Borrar todo">
  </form>

</body>

</html>
```

Cuando lo ejecutemos, deberíamos ver algo así:



Document x +

localhost:8080/practica1/test.php

NOMBRE

NIVEL DE INTERNET
bajo ☒ medio ☐ alto ☐

EXPERIENCIA PREVIA EN PROGRAMACIÓN
Visual Basic ☐ C/C++ ☐ Java ☐

TU OPINIÓN SOBRE ESTE CURSO

Enviar Borrar todo

2. ENVÍO Y RECEPCIÓN DE FORMULARIOS

2.1 Simple: con varias páginas web

Una vez enviado el formulario al servidor, éste debe recuperar la información para poder procesarla. Usamos el método GET, el cual tiene una variable especial donde se almacenan todas las variables que se pasen con este método. Se llama **\$_GET** y es un vector con índice *name* y contenido el valor que introduce el usuario en el formulario.

Del mismo modo, para recuperar la información enviada al servidor usamos el método POST. Éste también tiene una variable como en el caso anterior. Se llama **\$_POST**.

Así, para recibir los datos enviados con cualquiera de los dos métodos anteriores únicamente haremos uso de esas variables. PHP tiene una variable que contiene a las dos anteriores. Se trata de un vector asociativo llamado **\$_REQUEST**. Así, únicamente leyendo esta variable se podrá recuperar la información enviada usando ambos métodos.

- Se muestra el contenido de la matriz **\$_REQUEST** con la orden `print_r($_REQUEST)`
- Los índices de esta variable son los nombres de los controles
- El script de destino tendrá definida un variable por cada campo con el mismo **nombre que se le haya dado en el formulario. Y esta variable contendrá el valor que se haya introducido** en

el campo y está disponible desde el comienzo del script, no hay que hacer nada especial, sólo leer su contenido.

☐ Interesante

En ASP se ha de utilizar una variable distinta según el método de envío utilizado. Así, se emplea `request.QueryString()` en el caso de GET y `request.Form()` en el caso de POST.

En el caso de JSP, tanto para GET como para POST se usa `getParameter()`. Por tanto, almacena la información de la misma forma en los dos métodos, pero se envían de maneras distintas (GET en la URL y POST en el cuerpo del mensaje HTTP que el cliente envía al servidor).

Si seguimos desarrollando el ejemplo de la encuesta, vamos a diseñar una página PHP que reciba dichos datos y los muestre por pantalla. Vamos a guardarlo como *verificar.php*, que es el *action* que aparecía en el atributo del formulario de **encuesta.html**:

```
<?php
//verificar.php
echo "Comprueba si tus datos son correctos.<br>";
echo "<br>";
echo "{$_REQUEST["nombre"]}<br>";
echo "Nivel de internet: {$_REQUEST["nivel"]}<br>";

if (isset($_REQUEST["basic"]) || isset($_REQUEST["c_cplusplus"]) ||
isset($_REQUEST["java"])) {
    echo "Con experiencia en ";
    $experiencia = "";
    if (isset($_REQUEST["basic"])) {
        $experiencia = "Visual Basic";
        if (isset($_REQUEST["c_cplusplus"])) {
            $experiencia .= ", C++";
        }
        if (isset($_REQUEST["java"])) {
            $experiencia .= ", Java";
        }
    }
    elseif (isset($_REQUEST["c_cplusplus"])) {
        $experiencia = "C++";

        if (isset($_REQUEST["java"])) {
            $experiencia .= ", Java";
        }
    }
}
```

```
    }  
    } elseif (isset($_REQUEST["java"])) {  
        $experiencia = "Java";  
    }  
    echo "$experiencia.<br>";  
} else {  
    echo "Sin experiencia previa en programación.<br>";  
}  
echo "<br>";  
echo "OPINIÓN SOBRE EL CURSO:<br>";  
echo nl2br($_REQUEST["opinion"]); //convierte saltos de línea en <br>
```

Vamos a usar `$_REQUEST` para ver los parámetros que nos han llegado. Mostraremos su contenido con la función `print_r()`:

```
<pre>  
<?php  
print_r($_REQUEST);  
?>  
</pre>
```

2.2 Redirección de páginas web

En ocasiones se necesita que un programa PHP reciba datos y haga una operación "silenciosa" con los mismos. Es decir, que no sea necesario que se muestre una página al navegador sino que se vuelva a la inicial, como un formulario que envía, recibe y procesa datos.

Una alternativa es hacer una página que procese los datos y que al terminar se redirija automáticamente a otra. Para poder hacerlo PHP nos ofrece la función `header()`, que sirve para enviar cabeceras HTML, y que en el caso concreto que nos ocupa, se usa de la siguiente forma:

```
header("location:url")
```

La dirección de destino *url* puede ser una ruta absoluta o relativa.

Es imprescindible que la página no contenga al principio código *HTML*, ya que las cabeceras deben enviarse siempre al principio del documento. De hecho, aquí nos encontramos con un error muy habitual, ya que tan sólo con que se nos haya descuidado un espacio o salto de línea a enviar como *HTML* fallará. También hay que tener en cuenta que una vez que se llega a la función *header* no se sigue ejecutando el resto de la página, por lo que las operaciones que queramos hacer las deberemos ejecutar antes.

Veamos un ejemplo de recepción de datos. En primer lugar, se incluye una sencilla página web con un formulario que pide el nombre y un comentario para almacenarlo en un registro de visitas.

script *insertar.php* se almacena en la base de datos y se redirecciona a una página para dar las gracias.

- **visita.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Visita</title>
</head>
<body>
  <form name="visita" method="post" action="insertar.php">
    Nombre: <input type="text" name="nombre" size="44"><br>
    Comentario:<br>
    <textarea name="comentario" cols="40" rows="5"></textarea><br>
    <input type="submit" value="Enviar">
  </form>
</body>
</html>
```

- **insertar.php**

```
<?php
// Aquí vendría el código para guardar en la base de datos
header("location:gracias.html"); //Redirección
```

- **gracias.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Procesamiento solicitud</title>
</head>
<body>
  <p>Gracias por realizar la solicitud</p>
</body>
</html>
```

Si queremos redirigir a una página de error si hay fallos en los datos, deberemos validar una dirección de email y si es incorrecta derivarla a esa página de error:

- **suscripcion.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Suscripción al boletín electrónico</title>
</head>
<body>
  <p>Bienvenido al servicio de suscripción.<br>
    Por favor, introduzca una dirección de email válida.<br></p>

  <form name="suscripcion" method="post" action="guardar.php">
    Email: <input type="text" name="email" size="40"><br>
    <input type="submit" value="Enviar">
  </form>

</body>
</html>
```

- **guardar.php**

```
<?php
// Indica la posición del carácter "@" o FALSE si no está
$posicion_arroba = strpos($_REQUEST["email"], "@");
// Busca la aparición de un punto (.) a partir de la arroba
$posicion_punto = strpos($_REQUEST["email"], ".", $posicion_arroba);
if ($posicion_arroba && $posicion_punto) {
  // Aquí vendría el código para guardar en la base de datos
  header("location:confirmacion.html"); // Redirección
} else {
  // Aquí vendría el código para guardar en la base de datos
  header("location:error.html"); // Redirección
}
```

- **confirmacion.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Confirmación</title>
</head>
<body>
  <p>Su suscripción ha sido realizada con éxito. Gracias por confiar en
  nosotros</p>
</body>
</html>
```

- **error.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Confirmación</title>
</head>
<body>
  <p>Introdujo una dirección de correo no válida. Por favor vuelva a solicitar la
  suscripción</p>
  <a href="suscripcion.html"> Volver </a>
</body>
</html>
```

3. CONTROLES

En este anexo analizaremos los formularios con respecto al procesamiento de los datos.

Para que un control envíe información es necesario:

- Que el control esté incluido en un formulario (<form>)
- Que el formulario tenga establecido el atributo action, con la dirección absoluta o relativa del fichero php que procesará la información
- Que el control tenga establecido el atributo name
- Nota: el atributo *name* puede contener cualquier carácter (números, acentos, guiones, etc.), pero si contiene espacios, PHP sustituye los espacios por guiones bajos (_) al procesar los datos enviados
- Que el formulario contenga un botón de tipo submit
- Que contenga el método de envío (method). Si no lo contiene toma GET por defecto.

3.1 Botón enviar (submit button)

Este control se envía siempre. Se puede crear con la etiqueta `<input>` o con la etiqueta `<button>`. En esos casos se envía el valor del atributo *value* o el contenido de la etiqueta.

Código fuente	\$_REQUEST
<code><button type="submit">Submit</button></code>	Array ()
<code><button type="submit" name="boton3">Enviar</button></code>	Array ([boton3] => Enviar)
<code><button type="submit" name="boton4" value="enviado">Enviar</button></code>	Array ([boton4] => enviado)

3.2 Entradas de datos (input / textarea)

Este control se envía siempre. El valor enviado es el contenido de la caja o área.

Código fuente	Control	\$_REQUEST
<code><input type="text" name="cajatexto1" /></code>	<input type="text"/>	Array ([cajatexto1] =>)
<code><input type="text" name="cajatexto2" value="Cualquier cosa" /></code>	<input type="text" value="Cualquier cosa"/>	Array ([cajatexto2] => Cualquier cosa)
<code><input type="password" name="cajapassword1" /></code>	<input type="password"/>	Array ([cajapassword1] =>)
<code><input type="password" name="cajapassword2" value="pezespada" /></code>	<input type="password" value="pezespada"/>	Array ([cajapassword2] => pezespada)
<code><textarea rows="4" cols="20" name="areadetexto1"></textarea></code>	<div><div></div></div>	Array ([areadetexto1] =>)

3.3 Casilla de verificación (checkbox)

Este control se envía solamente si se marca la casilla. El valor enviado es "on" si la casilla no tiene definido el atributo *value*. En caso contrario se envía el valor del *value*.

Código fuente	Control	\$_REQUEST
<code><input type="checkbox" name="casilla1" /></code>	<input type="checkbox"/> (sin marcar)	Array ()
<code><input type="checkbox" name="casilla2" /></code>	<input checked="" type="checkbox"/> (marcada)	Array ([casilla2] => on)
<code><input type="checkbox" name="casilla3" value="Tres" /></code>	<input checked="" type="checkbox"/> (marcada)	Array ([casilla3] => Tres)

3.4 Botón radio

Este control se envía solamente si se marca alguno de los botones radio que forman el control. El valor enviado es "on" si el botón marcado no tiene definido el atributo value. En caso contrario se envía el valor de value.

<pre><input type="radio" name="radio1"> <input type="radio" name="radio1"></pre>	<input type="radio"/> (sin marcar)	Array ()
<pre><input type="radio" name="radio2"> <input type="radio" name="radio2"></pre>	<input checked="" type="radio"/> (marcado uno)	Array ([radio2] => on)
<pre><input type="radio" name="radio2"> <input type="radio" name="radio2"></pre>	<input type="radio"/> (marcado otro)	Array ([radio2] => on)
<pre><input type="radio" name="radio3" value="Uno"> <input type="radio" name="radio3" value="Dos"></pre>	<input checked="" type="radio"/> (marcado uno)	Array ([radio3] => Uno)
<pre><input type="radio" name="radio3" value="Uno"> <input type="radio" name="radio3" value="Dos"></pre>	<input type="radio"/> (marcado otro)	Array ([radio3] => Dos)

3.5 Menú

Este control envía siempre la opción elegida: el contenido de la etiqueta de dicha opción si no tiene definido el atributo *value*, o el valor de dicho atributo si sí existe. Si el menú admite selección múltiple, entonces el nombre del menú debe acabar con corchetes ([]) y se envía como una matriz, de tantos elementos como opciones se hayan elegido.

<pre><select name="menu1"> <option></option> </select></pre>	<input type="text"/>	Array ([menu1] =>)
<pre><select name="menu2"> <option>Opción 1</option> <option>Opción 2</option> </select></pre>	<input type="text" value="Opción 1"/>	Array ([menu2] => Opción 1)
<pre><select name="menu3"> <option value="Uno">Opción 1</option> <option value="Dos">Opción 2</option> </select></pre>	<input type="text" value="Opción 1"/>	Array ([menu3] => Uno)
<pre><select name="menu4[]" size="3" multiple> <option>Opción 1</option> <option>Opción 2</option> <option>Opción 3</option> <option>Opción 4</option> </select></pre>	<input type="text" value="Opción 1"/> <input type="text" value="Opción 2"/> <input type="text" value="Opción 3"/> (Marcados 1 y 3)	Array ([menu4] => Array ([0] => Opción 1 [1] => Opción 3))

3.6 Control oculto

Este control se envía siempre y manda el valor del atributo *value*.

<code><input type="hidden" name="oculto1"></code>		Array ([oculto1] =>)
<code><input type="hidden" name="oculto2" value="Cualquier cosa"></code>		Array ([oculto2] => Cualquier cosa)

3.7 Archivo

El selector de archivo permite enviar uno desde el ordenador del cliente al servidor.

Antes de usarlo hay que configurar en el archivo php.ini el tamaño máximo de los archivos que se pueden enviar (post_max_size)

En un formulario normal, este control se enviaría siempre y el valor enviado sería el nombre del archivo elegido:

Código fuente	Control	\$_REQUEST
<code><input type="file" name="archivo1" /></code>		Array ([archivo1] => loquesea.txt)

Para que este control envíe toda la información, el formulario debe tener el atributo **enctype** con el valor **multipart/form-data** y ser enviado por el método **POST**. La información será almacenada en la matriz **\$_FILES** y no en la variable **\$_REQUEST**:

Código fuente	\$_FILES
<code><form enctype="multipart/form-data" action="ejemplo.php" method="POST" > <input type="file" name="archivo2" /></code>	Array ([archivo2] => Array ([name] => loquesea.txt [type] => text/plain [tmp_name] => C:\ejemplos\loquesea.txt [error] => 0 [size] => 27890))

3.8 Imagen

El control de imagen inserta una imagen que funciona como un botón submit (no funciona en Firefox ni en Internet Explorer). Al pulsarlo envía las coordenadas del punto en el que se ha hecho clic, y si se define el atributo *value* también se envía el nombre del control con el valor del atributo:

Código fuente	Control	\$_REQUEST
<code><input type="image" name="GNU" alt="GNU" src="gnu.jpg" /></code>		Array ([GNU_x] => 89 [GNU_y] => 111)

4. CARGAR ARCHIVOS

Vamos a ver ahora cómo hacer que un formulario envíe un archivo al servidor y cómo hacer que éste lo procese.

4.1 Configuración de php (php.ini)

Para

poder recibir archivos, debes tener establecidas las siguientes directivas de configuración en el archivo PHP.ini:

- **file_uploads:** (On / Off), permite que haya o no cargas de archivos
- **upload_max_filesize:** tamaño máximo del archivo que se puede subir
- **upload_tmp_dir:** directorio temporal donde se guardan los archivos cargados
- **post_max_size:** tamaño máximo de los datos enviados por el método post

4.2 Configuración del formulario

Un formulario puede enviar un archivo al servidor mediante un control de tipo file. La sintaxis es:

```
<input type="file" size=" " name=" " >
```

El formulario sería:

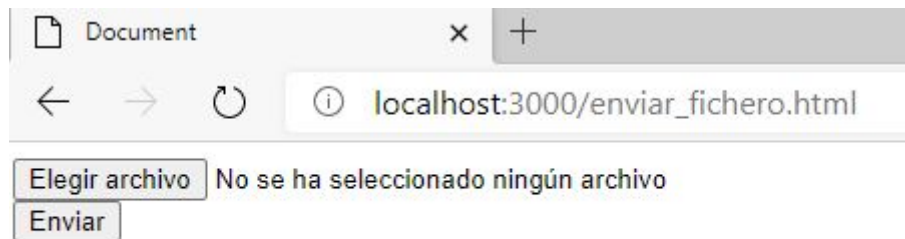
```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <form action="carga_archivos.php" method="post"
  enctype="multipart/form-data">
    <input type="file" name="nombre_archivo_cliente" /><br />
    <input type="submit" name="enviar" value="Enviar" />
  </form>
</body>
</html>
```

Es importante que el atributo *method* tenga el valor post y que el atributo *enctype* tenga el valor multipart/form-data. Recuerda que el fichero tiene límite en tamaño, que puede ser fijado desde php.ini (upload_max_size) o en el propio formulario tal y como se muestra:

```
<input type="hidden" name="max_file_size" value="102400" >
<input type="file" name="enviar">
```

Es importante destacar que debe llamarse max_file_size y que su valor es un entero que indica el máximo de bytes.

El aspecto del formulario dependerá del navegador que utilices. En este navegador sería:



4.3 Tratamiento de ficheros en php

Cuando PHP recibe el archivo, lo almacena en el directorio `upload_tmp_dir` y rellena la matriz asociativa superglobal `$_FILES["nombre_archivo"]` (el nombre que se haya dado al control en el formulario). Esa matriz contiene los siguientes elementos:

Nombre	Descripción
<code>\$_FILES['nombre_archivo_cliente']['name']</code>	nombre que tenía el archivo cargado en el ordenador del cliente
<code>\$_FILES['nombre_archivo_cliente']['type']</code>	tipo MIME del archivo cargado
<code>\$_FILES['nombre_archivo_cliente']['size']</code>	tamaño del archivo cargado
<code>\$_FILES['nombre_archivo_cliente']['tmp_name']</code>	nombre del archivo cargado en el directorio temporal del servidor
<code>\$_FILES['nombre_archivo_cliente']['error']</code>	código de error (en su caso)

Debe darse al fichero un nombre único. De este modo, se suele descartar el nombre original del fichero y se crea uno nuevo que sea único (por ejemplo añadiéndole la fecha y la hora).

Es importante recordar que el fichero subido se almacena en un directorio temporal, por lo que desaparecerá al acabar el script. Es, por tanto, necesario copiar el archivo a otro lugar. Para ello utiliza la función `move_uploaded_file($origen, $destino)`, en la que **\$origen** es el nombre del archivo cargado en el directorio temporal (normalmente puedes utilizar directamente `$_FILES['nombre_archivo']['tmp_name']`) y **\$destino** el nombre del archivo que contendrá la copia.

Debemos realizar varias operaciones o comprobaciones antes de copiar el fichero en su ubicación destino:

Veamos un ejemplo:

- **enviar_fichero.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <form action="carga_archivos.php" method="post" enctype="multipart/form-data">
    <input type="file" name="imagen" /><br />
    <input type="submit" name="enviar" value="Enviar" />
  </form>
</body>
</html>
```

- **carga_archivos.php**

```
<?php
$idUnico = rand(0, 46598461654);
if (is_uploaded_file($_FILES['imagen']['tmp_name'])) {
  $nombreDirectorio = "img/";
  $nombreFichero = $_FILES['imagen']['name'];
  $nombreFichero = $idUnico . "-" . $nombreFichero;
  move_uploaded_file($_FILES['imagen']['tmp_name'], $nombreDirectorio .
  $nombreFichero);
} else {
  print("No se ha podido subir el fichero\n");
}
```

Analicemos el ejemplo: `is_uploaded_file` devuelve TRUE si el archivo que le pasamos se ha subido por HTTP POST. Con esto se evita que el usuario intente usar archivos del servidor (/etc/passwd). Una vez comprobado que hemos subido correctamente el archivo, asignamos un nuevo nombre al fichero añadiéndole la marca de tiempo (`tmp_name`). Tras esto, ya podemos mover el fichero a su ubicación definitiva con `move_uploaded_file`. Si no pudiera moverlo daría error.

Interesante

Puedes ver más información de estas funciones accediendo a los manuales siguientes:

- * is_uploaded_file: <http://php.net/manual/es/function.is-uploaded-file.php>
- * is_file: <http://php.net/manual/es/function.is-file.php>
- * move_uploaded_file: <http://php.net/manual/es/function.move-uploaded-file.php>

5. RECOGIDA DE DATOS

Como ya hemos visto, cuando se envía un formulario, PHP almacena la información en la matriz `$_REQUEST`.

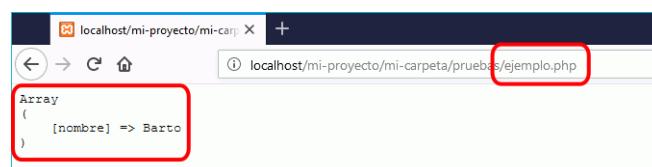
Cualquier control se envía solamente si está así establecido en su atributo *name*. Éste puede contener cualquier carácter (letras, números, acentos, guiones...) pero si contiene espacios se sustituyen por guiones bajos "_". Cada control crea un elemento de la matriz `$_REQUEST`, que se identifica como `$_REQUEST[valor_del_atributo_name]` y que contiene el valor entregado por el formulario si corresponde.

Veamos un ejemplo:

```
<form action="ejemplo.php">
<p>Nombre: <input type="text" name="nombre"/></p>
<p><input type="submit" value="Enviar" /></p>
</form>
```

Mientras programamos, la manera más fácil de comprobar que nuestro fichero PHP está recibiendo la información enviada por el control es usar la función **`print_r($matriz)`**. Ésta muestra el contenido de `$_REQUEST`. Pero no debemos olvidar borrar esta línea de código una vez comprobado.

Veamos qué ocurre si el programa PHP recibe la información del formulario anterior:



Es recomendable colocar las etiquetas **`<pre>`** alrededor del `print_r($_REQUEST)` para facilitar la visualización de los valores.

Cuando hacemos referencia a los elementos de `$_REQUEST` debemos tener en cuenta que si lo hacemos desde dentro de una cadena, al control de nombre "nombre" debemos escribirle `$_REQUEST[nombre]` sin comillas.

Y si lo hacemos la referencia desde fuera de una cadena al control de nombre "nombre", debemos usar comillas:

<pre><?php print "<p>Su nombre es \$_REQUEST['nombre']</p>"; ?></pre>	Parse error: parse error, unexpected T_ENCAPSED_AND_WHITESPACE, expecting T_STRING or T_VARIABLE or T_NUM_STRING in ejemplo.php on line 2
<pre><?php print "<p>Su nombre es \$_REQUEST[nombre]</p>"; ?></pre>	Su nombre es Pepe

5.1 Comprobación de existencia

- **isset(\$variable)**

La función `isset()` determina si una variable está definida y no es `NULL`. Su sintaxis es:

```
isset ( variables_separadas_por_comas ) : bool
```

Más información en: <https://www.php.net/manual/es/function.isset.php>

La mayoría de los controles aparecen en la matriz `$_REQUEST` aunque el usuario no escriba nada en el formulario (en ese caso, se almacena como una cadena vacía). Con tal de evitar errores, se recomienda usar una estructura `IF..ELSE` para tener en cuenta que no se haya escrito nada en el formulario:


```
1 if (isset($_REQUEST['nombre'])){
2     $nombre = $_REQUEST['nombre']
3 } else {
4     $nombre = "";
5 }
```

Sin embargo, **hay controles que sólo aparecerán en la matriz \$_REQUEST si se marca en el formulario**. Es el caso de las **casillas de verificación y los botones radio**. Veamos un ejemplo donde el usuario no marca la casilla y así la matriz no contiene datos:

<pre><form action="ejemplo.php"> <p>Deseo recibir información: <input type="checkbox" name="acepto" /></p> <p><input type="submit" value="Enviar" /></p> </form></pre>	Deseo recibir información: <input type="checkbox"/> Enviar	
Deseo recibir información: <input type="checkbox"/> Enviar	<pre><?php print "<pre>"; print r(\$_REQUEST); print "</pre>\n"; ?></pre>	Array ()

Y al usar el valor da un error.

Esto se resuelve comprobando que el índice está definido antes de hacer referencia a él. Para ello, debemos emplear la función **isset(\$variable)**, que admite como argumento una variable y devuelve TRUE si existe y FALSE si no.

```
1  if (isset($_REQUEST['nombre'])){  
2      $nombre = $_REQUEST['nombre']  
3  } else {  
4      $nombre = "";  
5  }
```

En conclusión: siempre deberíamos verificar que exista el índice.

6. SEGURIDAD EN LAS ENTRADAS

6.1 strip_tags

Se aplica a las versiones PHP 4, PHP 5, PHP 7. Strip_tags retira las etiquetas HTML y PHP de un string.

Ejemplo:

```
<!DOCTYPE html>  
<html>  
<body>  
    <?php  
        echo strip_tags("Hello <b>world!</b>");  
    ?>  
</body>  
</html>
```

Tiene como resultado: Hello world!

Más información en: <https://www.php.net/manual/es/function.strip-tags.php>

Se puede probar en php en:

https://www.w3schools.com/php/phptryit.asp?filename=tryphp_func_string_strip_tags

Un usuario puede insertar código HTML en la entrada de un formulario, lo que puede acarrear comportamientos inesperados y riesgos de seguridad. El siguiente ejemplo solamente perjudicaría al aspecto de la página:

<p>Nombre: <input type="text"/></p> <p>Enviar</p>	<pre><?php print "<pre>"; print_r(\$_REQUEST); print "</pre>\n"; print "<p>Su nombre es \$_REQUEST[nombre]</p>"; ?></pre>	<pre>Array ([nombre] => Pepe)</pre> <p>Su nombre es Pepe</p>
---	--	---

Veamos cómo usar un formulario para introducir código Javascript. En este caso, se genera una ventana emergente que sólo afecta al usuario pero que podría ser usada para atacar al servidor:

<p>Nombre: <input type="text"/></p> <p>Enviar</p>	<pre><?php print "<pre>"; print_r(\$_REQUEST); print "</pre>\n"; print "<p>Su nombre es \$_REQUEST[nombre]</p>"; ?></pre>	<div style="border: 1px solid black; padding: 10px; text-align: center;"> <p>HOLA</p> <p>Aceptar</p> </div>
---	--	---

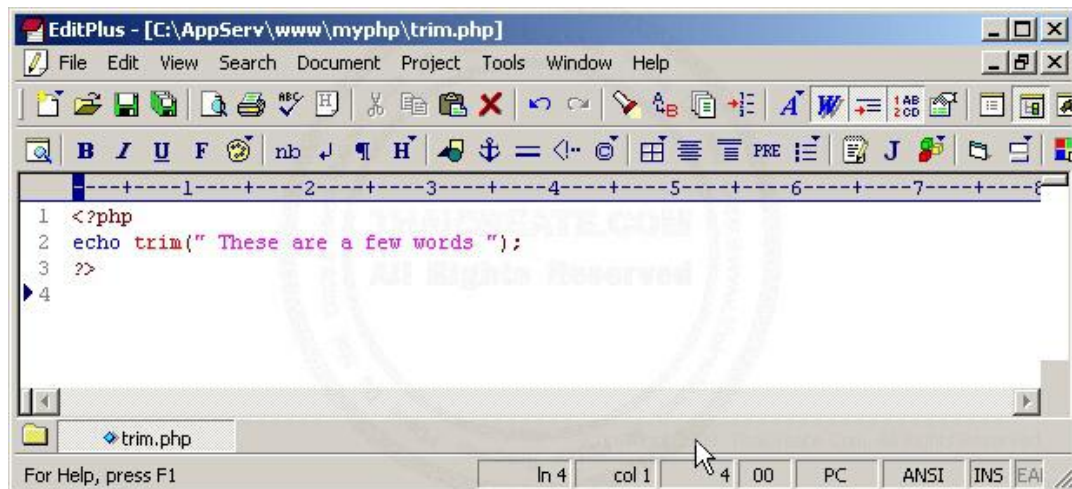
Para evitarlo, usaremos la función **strip_tags(\$cadena)**, que elimina todas las etiquetas HTML:

<p>Nombre: <input type="text" value="Pepe"/></p> <p>Enviar</p>	<pre><?php print "<pre>"; print_r(\$_REQUEST); print "</pre>\n"; print "<p>Su nombre es" . strip_tags(\$_REQUEST['nombre']) . "</p>"; ?></pre>	<pre>Array ([nombre] => Pepe)</pre> <p>Su nombre es Pepe</p>
--	---	---

El problema es que esta función elimina también cualquier cosa que interprete como etiqueta, así que no deberemos usarla si queremos conservar algún carácter como por ejemplo "<".

6.2 trim

Otra función que debemos aplicar en cualquier entrada de formulario es **trim(\$cadena)**. Ésta elimina los espacios en blanco iniciales y finales de la cadena y la devuelve sin ellos. Veamos un ejemplo de buen funcionamiento:



The screenshot shows the EditPlus text editor with the following PHP code:

```
1 <?php
2 echo trim(" These are a few words ");
3 ?>
4
```

The status bar at the bottom indicates the cursor is at line 4, column 1.

6.3 Uso de variables

Tomando en cuenta las medidas de seguridad antes comentadas, se deduce que cualquier referencia a la matriz `$_REQUEST[control]` debería hacerse con la función `trim(strip_tags($_REQUEST[control]))`. Pero se complica demasiado cuando además queremos hacer un `isset()` para comprobar que está definido dicho código.

Una solución elegante es guardar los valores de la matriz en variables y hacer todas las comprobaciones al definir dichas variables. Así, podemos usarlas a ellas en vez de a la matriz:

<p>Nombre:</p> <input type="text"/> <input type="button" value="Enviar"/>	<pre><?php print "<pre>"; print_r(\$_REQUEST); print "</pre>\n"; if (isset(\$_REQUEST['nombre'])) { \$nombre = trim(strip_tags(\$_REQUEST['nombre'])); } else { \$nombre = ""; } if (\$nombre == "") { print "<p>No ha escrito ningún nombre</p>"; } else { print "<p>Su nombre es \$nombre</p>"; } ?></pre>	<pre>Array ([nombre] =>)</pre> <p>No ha escrito ningún nombre</p>
<p>Nombre:</p> <input type="text" value="Pepe"/> <input type="button" value="Enviar"/>	<pre><?php print "<pre>"; print_r(\$_REQUEST); print "</pre>\n"; if (isset(\$_REQUEST['nombre'])) { \$nombre = trim(strip_tags(\$_REQUEST['nombre'])); } else { \$nombre = ""; } if (\$nombre == "") { print "<p>No ha escrito ningún nombre</p>"; } else { print "<p>Su nombre es \$nombre</p>"; } ?></pre>	<pre>Array ([nombre] => Pepe)</pre> <p>Su nombre es Pepe</p>

Y si queremos hacer la asignación de la variable en una sola línea, podemos usar la notación siguiente: *(condición) ? verdadero : falso;*

6.4 Salida de datos

- **htmlspecialchars()**

Su utiliza en las versiones PHP 4, PHP 5, PHP 7. Htmlespecialchars convierte caracteres especiales en entidades HTML.

Más información en: <https://www.php.net/manual/es/function htmlspecialchars.php>

Debemos tener ciertas precauciones si los datos que se recogen se van a escribir luego en la página web. Veamos dos casos y luego la solución:

Caso 1: el carácter &

Si el usuario escribe en una entrada el carácter & (ampersand, entidad de carácter &#amp;#38;), si esa cadena se escribe en una página, la página se verá correctamente en el navegador, pero la página no será válida (el xhtml será invalido). Esto es porque dicho carácter indica el comienzo de una entidad de carácter. Veamos un ejemplo de error de tipo:

<p>Nombre:</p> <input type="text" value="Pepito & Cor"/> <input type="button" value="Enviar"/>	<pre><?php \$nombre = (isset(\$_REQUEST['nombre'])) ? trim(strip_tags(\$_REQUEST['nombre'])) : ""; if (\$nombre == "") { print "<p>No ha escrito ningún nombre</p>"; } else { print "<p>Su nombre es \$nombre</p>"; } ?></pre>	<p>Su nombre es Pepito & Company</p>
---	---	--

Caso 2: el carácter "

Si el usuario escribe en una entrada el carácter " (comillas, entidad de carácter "), si esa cadena se escribe dentro de otras comillas (por ejemplo, en el atributo *value* de una etiqueta *input*), la página no se verá correctamente y además no será válida. Veamos un ejemplo de un error de este tipo:

Nombre: <input type="text" value="Me llamo 'Pepe'"/> <input type="button" value="Enviar"/>	<pre><?php \$nombre = (isset(\$_REQUEST["nombre"])) ? trim(strip_tags(\$_REQUEST["nombre"])) : ""; if (\$nombre == "") { print "<p>No ha escrito ningún nombre</p>\n"; } else { print "Corrija: <input type='text' value='\"\$nombre\"'></p>\n"; } ?></pre>	Corrija: <input type="text" value="Me llamo"/>
---	---	--

Solución general

Existen más casos (el uso de ', <, >, etc.) En todos los casos se podría sustituir dicho carácter por su entidad de carácter correspondiente a través de la función **str_replace()** (ejemplo `$nombre = str_replace("'", '"', $nombre)` para las comillas o `$nombre = str_replace('& ', '& ', $nombre)` para el &). Pero esto nos obligaría a hacerlo para cada una de ellas, así que lo más útil es usar la función **htmlspecialchars()** que lo hace de una vez todo. Eso sí, si usamos la función **strip_tags()** previamente podría darse el caso de que eliminara etiquetas que queremos conservar, así que deberíamos invertir el orden. Veamos un ejemplo correcto:

Nombre: <input type="text" value="Me llamo 'Pepe'"/> <input type="button" value="Enviar"/>	<pre><?php \$nombre = (isset(\$_REQUEST["nombre"])) ? trim(strip_tags(\$_REQUEST["nombre"])) : ""; \$nombre = str_replace("'", '&quot;', \$nombre); if (\$nombre == "") { print "<p>No ha escrito ningún nombre</p>\n"; } else { print "Corrija: <input type='text' value='\"\$nombre\"'></p>\n"; } ?></pre>	Corrija: <input type="text" value="Me llamo 'Pepe'"/>
---	--	---

6.5 Definir funciones de recogida de datos

Tener en cuenta todos los aspectos vistos es complicado. Es por ello que lo recomendable es definir una función para recoger los datos, y así poder desentendernos a posteriori de este asunto. Veamos un ejemplo:

```
<?php
// FUNCIÓN DE RECOGIDA DE UN DATO
function recoge($var)
{
    $tmp = (isset($_REQUEST[$var])) ?
```

```
strip_tags(trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES,
"ISO-8859-1"))): "");
    if (get_magic_quotes_gpc()) {
        $tmp = stripslashes($tmp);
    }
    return $tmp;
}
```

```
// EJEMPLO DE USO DE LA FUNCIÓN ANTERIOR
$nombre = recoge('nombre');
if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>";
} else {
    print "<p>Su nombre es $nombre</p>";
}
?>
```

De ella se deducen varias cosas:

- La función *recoge()* del ejemplo tiene como argumento el nombre del control que se quiere recibir y devuelve el valor recibido o una cadena vacía si el control no se ha recibido.
- Además, elimina las etiquetas que pueda contener el texto. Si se desearan recoger etiquetas, la función tendría que ser distinta.
- Al comienzo del programa el dato recibido se almacena en una variable.
- En el resto del programa se trabaja con la variable.

Si quisiéramos recortar el tamaño de los datos recibidos podríamos crear una función *recorta()* que recortara la longitud de los campos y que fuera llamada desde la función *recoge()* que creado en el ejemplo anterior:

```
<?php
// FUNCIONES DE RECOGIDA Y RECORTE DE UN DATO
$tamNombre = 30;

$recorta = array(
    'nombre' => $tamNombre);

function recorta($campo, $cadena)
{
    global $recorta;
    $tmp = isset($recorta[$campo])? substr($cadena, 0, $recorta[$campo]):
    $cadena; return $tmp;
```



```
}

function recoge($var)
{
    $tmp = (isset($_REQUEST[$var]))?
    strip_tags(trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES,
    "ISO-8859-1")))) : "";
    if (get_magic_quotes_gpc()) {
        $tmp = stripslashes($tmp);
    }
    $tmp = recorta($var, $tmp); return $tmp;
}

// EJEMPLO DE USO DE LAS FUNCIONES ANTERIORES
$nombre = recoge('nombre');
if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>";
} else {
    print "<p>Su nombre es $nombre</p>";
}
?>
```

7. COMPROBACIÓN DE DATOS CON FUNCIONES

Una vez recibido un dato normalmente es necesario comprobar si el dato es del tipo esperado (número, texto, etc.) para procesarlo sin error. PHP ofrece varios conjuntos de funciones para realizar estas comprobaciones. Vamos a ver alguna de ellas. *NOTA: puede que alguna función de las que se describen aquí no existan en versiones anteriores, así que antes de usarlas deberemos asegurarnos de que la tenemos.*

Seguidamente vamos a ver las Funciones `xxx_exists()` para comprobación de datos..

7.1 Función `_exists()`

Es una función booleana que comprueba si una función existe o no. Puedes ver más detalles en <http://php.net/manual/es/function.function-exists.php>

Veamos un ejemplo:

```
<?php
var_dump(filter_var('bob@example.com', FILTER_VALIDATE_EMAIL));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
?>
```

El resultado del ejemplo sería:

```
string(15) "bob@example.com"
bool(false)
```

Donde *filter_var()* es una función que valida una variable con el filtro que se indique.

7.2 Función array_key_exists()

Es una función booleana con la sintaxis `array_key_exists($indice,$matriz)`. Devuelve si un elemento determinado de una matriz existe o no. Para esto también se puede usar la función *isset()* que veremos a continuación.

7.3 Funciones is_()

Son un conjunto de funciones booleanas que devuelven si el argumento es o no de un tipo de datos determinado. En la siguiente tabla se encuentran su mayoría:

Tipo	Función	Tipo de datos	alias (funciones equivalentes)
existencia	isset(\$valor)	definida	
	is_null(\$valor)	NULL	
números	is_bool(\$valor)	booleano	
	is_numeric(\$valor)	número (puede tener signo, parte decimal y estar expresado en notación decimal, exponencial o hexadecimal).	
	is_int(\$valor)	entero	is_integer(\$valor) , is_long(\$valor)
	is_float(\$valor)	float	is_double(\$valor) , is_real(\$valor)
cadenas	is_string(\$valor)	cadena	
otros	is_scalar(\$valor)	escalar (entero, float, cadena o booleano)	
	is_array(\$valor)	matriz	

	is_callable(\$valor)	función	
	is_object(\$valor)	object	
	is_resource(\$valor)	recurso	

Debemos tener en cuenta que si un dato es demasiado grande para el tipo de variable, las funciones devolverán FALSE. Por ejemplo, si se escribe un entero mayor que PHP_INT_MAX(2147483647 en Windows) la función *is_int()* devolverá FALSE. Las funciones *is_int()* e *is_float()* devuelven FALSE con los valores provenientes de un formulario, ya que se reciben como cadenas y no como variables de tipo numérico.

Vamos a ver el funcionamiento de una de ellas: *is_numeric()*

7.4 Funciones ctype_()

Son un conjunto de funciones booleanas que devuelven si todos los caracteres de una cadena son de un tipo determinado, de acuerdo con el juego de caracteres local. Estas funciones son las mismas que las que proporciona la biblioteca estándar de C *ctype.h*.

Función	Tipo de datos
ctype_alnum(\$valor)	alfanuméricos
ctype_alpha(\$valor)	alfabéticos (mayúsculas o minúsculas, con acentos, ñ, ç, etc.)
ctype_cntrl(\$valor)	caracteres de control (salto de línea, tabulador, etc.)
ctype_digit(\$valor)	dígitos
ctype_graph(\$valor)	caracteres imprimibles (excepto espacios)
ctype_lower(\$valor)	minúsculas
ctype_print(\$valor)	caracteres imprimibles
ctype_punct(\$valor)	signos de puntuación (caracteres imprimibles que no son alfanuméricos ni espacios en blanco)
ctype_space(\$valor)	espacios en blanco (espacios, tabuladores, saltos de línea, etc.)
ctype_upper(\$valor)	mayúsculas
ctype_xdigit(\$valor)	dígitos hexadecimales

Veamos como ejemplo cómo funciona *ctype_digit()*:

```
<?php
$cadenas = array('1820.20', '10002', 'wsl!12');
foreach ($cadenas as $caso_prueba) {
    if (ctype_digit($caso_prueba)) {
        echo "La cadena $caso_prueba consiste completamente de dígitos.\n";
    } else {
        echo "La cadena $caso_prueba no consiste completamente de dígitos.\n";
    }
}
?>
```

El resultado del ejemplo sería:

```
La cadena 1820.20 no consiste completamente de dígitos.
La cadena 10002 consiste completamente de dígitos.
La cadena wsl!12 no consiste completamente de dígitos.
```

7.5 Funciones filter

Se crearon como extensión PECL, pero ya forman parte de PHP desde la versión 5.2 y ni siquiera es necesario activarlas en php.ini. Puedes ver más información sobre ellas en <http://www.php.net/manual/es/book.filter.php>

La más simple es la función `filter_var($valor [, $filtro [, $opciones]])`, que devuelve los datos filtrados o FALSE si el filtro falla. Los filtros predefinidos son los siguientes:

Filtro	Tipo de datos
<code>FILTER_VALIDATE_INT</code>	entero
<code>FILTER_VALIDATE_BOOLEAN</code>	booleano
<code>FILTER_VALIDATE_FLOAT</code>	float
<code>FILTER_VALIDATE_REGEXP</code>	expresión regular
<code>FILTER_VALIDATE_URL</code>	URL
<code>FILTER_VALIDATE_EMAIL</code>	dirección de correo
<code>FILTER_VALIDATE_IP</code>	IP

8. COMPROBACIÓN DE DATOS CON EXPRESIONES REGULARES

Las expresiones regulares permiten definir patrones de coincidencia y aplicarlas a cadenas de texto para saber si la cadena (o parte de ella) cumple el patrón e incluso realizar transformaciones de la cadena.

En PHP existen dos sistemas de expresiones regulares:

- **Expresiones regulares POSIX extendido:**
 - fueron introducidas en PHP 2.0b7 en 1996, y consideradas obsoletas a partir de PHP 5.3.0 (junio de 2009). Siguen la recomendación POSIX 1003.2. POSIX (Portable Operating System Interface) es un conjunto de normas redactadas por el IEEE (The Institute of Electrical and Electronics Engineers) que definen la API de Unix. Las funciones correspondientes empiezan por "ereg".
- **Expresiones regulares compatibles con Perl (en inglés, PCRE):**
 - fueron introducidas en PHP 4.2.0 en 2002. Siguen la sintaxis y semánticas del lenguaje de programación Perl 5. PHP 4.2.0 y posteriores incluyen la biblioteca de código libre escrita en C PCRE (Perl Compatible Regular Expressions). Las funciones correspondientes empiezan por "preg".

Dado que las funciones **ereg (POSIX extendido) se consideran obsoletas** a partir de PHP 5.3.0 (publicado en junio de 2009), se recomienda utilizar únicamente las funciones preg (PCRE). Son las que vamos a detallar a continuación.

En la siguiente tabla se lista un resumen de las más relevantes:

8.1 Funciones de comparación

La función de expresiones regulares compatibles con Perl compara una cadena con un patrón y devuelve 1 si el patrón ha coincidido o 0 si no. Los patrones deben empezar y acabar con el carácter / (barra). Un ejemplo sería:

```
<?php
$cadena1 = "1234567";
$cadena2 = "abcdefg";
$patron = "/^[[[:digit:]]+$/";

if (preg_match($patron, $cadena1)) {
    print "<p>La cadena $cadena1 son sólo números.</p>\n";
} else {
    print "<p>La cadena $cadena1 no son sólo números.</p>\n";
}

if (preg_match($patron, $cadena2)) {
    print "<p>La cadena $cadena2 son sólo números.</p>\n";
} else {
```

```
print "<p>La cadena $cadena2 no son sólo números.</p>\n";
}
```

Devuelve por pantalla: “La cadena 1234567 no son sólo números. La cadena abcdefg no son sólo números”.

En este ejemplo vemos el uso de la función **preg_match()**. Ésta distingue entre mayúsculas y minúsculas, y si no queremos que lo haga debe añadirse el modificador **i** al final del patrón:

```
<?php
$cadena = "aaAA";
$patron1 = "/^[a-z]+$/";
$patron2 = "/^[a-z]+$/i";

if (preg_match($patron1, $cadena)) {
    print "<p>La cadena $cadena son sólo letras minúsculas.</p>\n";
} else {
    print "<p>La cadena $cadena no son sólo letras minúsculas.</p>\n";
}

if (preg_match($patron2, $cadena)) {
    print "<p>La cadena $cadena son sólo letras minúsculas o mayúsculas.</p>\n";
} else {
    print "<p>La cadena $cadena no son sólo letras minúsculas o mayúsculas.</p>\n";
}
?>
```

Devuelve por pantalla “La cadena aaAA no son sólo letras minúsculas. La cadena aaAA son sólo letras minúsculas o mayúsculas”.

La función **preg_match_all()** compara una cadena con un patrón y devuelve el número de coincidencias encontradas. Éste puede guardarse en un argumento opcional, **\$matriz_coincidencias**, y si se añade el modificador **PREG_OFFSET_CAPTURE** también se almacenan las posiciones de cada coincidencia encontrada. El argumento opcional **\$desplazamiento** es un número que permite indicar en qué carácter se inicia la búsqueda. Veamos un ejemplo:

```
<?php
$cadena = "Esto es una cadena de prueba";
$patron = "/de/";
$encontrado = preg_match_all($patron, $cadena,
    $coincidencias, PREG_OFFSET_CAPTURE);

if ($encontrado) {
    print "<pre>"; print_r($coincidencias); print "</pre>\n";
    print "<p>Se han encontrado $encontrado coincidencias.</p>\n"; foreach ($coincidencias[0]
    as $coincide) {
        print "<p>Cadena: '$coincide[0]' - Posición: $coincide[1]</p>\n";
    }
}
```

```
} else {  
print "<p>No se han encontrado coincidencias.</p>\n";  
}  
?>
```

Devuelve por pantalla “Se han encontrado 2 coincidencias. Cadena: ‘de’-Posición: 14. Cadena: ‘de’-Posición: 19”

8.2 Funciones de búsqueda

La función es **preg_replace**(patron, sustituta, cadena, limite_coincidencias, \$contador). En ella:

- Patron: es el patrón de búsqueda
- Sustituta: es la cadena que va a sustituir las coincidencias
- Cadena: es la cadena en la que pretendemos realizar la búsqueda y sustitución
- limite_coincidencias: es el número máximo de coincidencias a las que aplicaremos el reemplazamiento
- \$contador: es el nombre de una variable que puede utilizarse como contador del número de cambios efectuados.

Si se omiten los parámetros *limite_coincidencias* y *\$contador* se reemplazarán todos los valores posibles sin recoger en ninguna variable el número de cambios

El parámetro patrón puede ser un array en el que cada elemento será un patrón de búsqueda. Cuando se da esa circunstancia y sustituta es una cadena, las coincidencias de cada uno de los diferentes patrones son reemplazadas por la cadena sustituta. En el caso de que sustituta fuera un array, las coincidencias del patrón contenido en cada elemento de array serán reemplazadas por el valor del elemento de sustituta de igual índice.

Veamos un ejemplo:

```
<?php  
$cadena = 'El veloz murciélago hindú comía feliz cardillo y kiwi.'; echo "Antes de la  
sustitución <br> $cadena <br>";  
$patrones = array();  
$patrones[0] = '/veloz/';  
$patrones[1] = '/hindú/';  
$patrones[2] = '/murciélago/';  
$sustituciones = array();  
$sustituciones[2] = 'galápagos';  
$sustituciones[1] = 'africano';  
$sustituciones[0] = 'lento';  
echo "Después de la sustitución <br>";  
echo preg_replace($patrones, $sustituciones, $cadena);  
?>
```

8.3 Búsqueda con

metacaracteres

Llamaremos metacaracteres a un conjunto de caracteres que tienen un significado especial cuando se incluyen en una expresión regular. A título puramente enumerativo, forman parte de ese conjunto los siguientes: ., +, *, {, }, [,], -, \, ^, \$, |, (,) y ?.

Metacarácter	Descripción
\	escape
^	inicio de string o línea
\$	final de string o línea
.	coincide con cualquier carácter excepto nueva línea
[inicio de la definición de clase carácter
]	fin de la definición de clase carácter
	inicio de la rama alternativa
(inicio de subpatrón
)	fin de subpatrón
?	amplia el significado del subpatrón, cuantificador 0 ó 1, y hace lazy los cuantificadores greedy
*	cuantificador 0 o más
+	cuantificador 1 o más
{	inicio de cuantificador mín/máx
}	fin de cuantificador mín/máx

El gran problema de los metacaracteres es querer incluirlos como un carácter más en el patrón de búsqueda. Para poder hacerlo han de “pasar de metacaracteres a caracteres normales que serán interpretados como texto” y a eso se le llama “escaparlos”. Esto se hace anteponiéndoles una barra invertida delante “\”.

La función `$r= preg_quote($c, $d)` recoge en la variable “\$r” el resultado de aplicar esa secuencia de escape a los caracteres especiales en expresiones regulares, la cadena contenida como parámetro en la variable “\$c”. El parámetro “\$d” deberá recoger el carácter delimitador del patrón de búsqueda.


```
<?php
$claves = '$40 por un a g3/400';
$claves = preg_quote($claves, '/');
echo $claves; // devuelve \$40 por un g3\400
?>
```

8.4 Búsqueda con alternancias

La alternancia utiliza el carácter “|” para permitir una elección entre dos o más alternativas. Cuando se conjugan incluidas entre paréntesis y teniendo en cuenta que los paréntesis pueden agruparse y/o anidarse pueden construirse expresiones tan complejas como necesitemos.

```
let regexp = /html|php|css|java(script)?/gi;

let str = "Primera aparición de HTML, luego CSS, luego JavaScript";
```

8.5 Búsqueda con anclas

En los ejemplos que hemos visto hasta ahora las coincidencias podrían producirse en cualquier parte de la cadena en la que efectuamos la búsqueda. Estas condiciones pueden modificarse restringiendo la búsqueda al principio o al final de la cadena de búsqueda. Para el primer caso se antepone al patrón el metacarácter “^”.

Cuando se trata de buscar coincidencias únicamente al final de la cadena es necesario recurrir al metacarácter “\$” que debe ser incluido detrás del patrón de búsqueda. Tanto este metacarácter

como “^” cambian su funcionalidad cuando se incluye el modificador “m”. Cuando se da esta situación considerará comienzo y final tanto el comienzo y el final de la cadena como las posiciones anteriores y posteriores al carácter de salto de línea “\n”.

9. BIBLIOGRAFÍA

Libros

- [CIB12] Cibelli, C. (2012): *PHP. Programación web avanzada para profesionales*, Marcombo - Alfaomega, Barcelona
- [DUR13] Duran, C. (2013): *Recuperación y utilización de la información proveniente del cliente web*
- [LOP12] López, M.; Vara, JM; Verde, J.; Sánchez, D.M.; Jiménez, J.J.; Castro, V. (2012): *Desarrollo web en entorno servidor*, RA-MA, Madrid

Webs

- Validación de formulario php.
http://www.ahoracapacitaciones.com/cursos/php5/scripts/leccion_6/code_tester.php?var=code_03