[**Recordatorio GET y POST**](#_udeussnqtu32) **1**

[**Validación de datos del formulario en cliente**](#_x2o9hfi2sg1x) **2**

[Validaciones con HTML5](#_qfw3sdchhgqm) 4

[**Enviando y recibiendo información**](#_v7mmqaaxgkkt) **6**

[**Validación de datos en el servidor (PHP)**](#_mkg9mk7gw8wn) **7**

[**Subir archivos con php**](#_kzm61jd0epob) **9**

[Subida múltiple de archivos](#_uesv03w3quft) 12

Muchas aplicaciones están diseñadas para formular preguntas que los usuario responden digitando información. A veces, la información se almacena en una base de datos; otras, la información se usa en enunciados condicionales para entregar una página web individual. Algunas de las tareas más comunes de las aplicaciones que requieren que los usuarios respondan preguntas son:

* Hacer pedidos en línea
* Inscribirse
* Registrarse
* Ver información seleccionada

Las preguntas se formulan mediante formularios HTML. El usuario responde las preguntas digitando información en el formulario o seleccionando elementos de una lista. El usuario luego hace clic en un botón para enviar la información del formulario. Cuando el formulario es enviado, la información en él se pasa a un segundo programa separado, el cual procesa la información.

## Recordatorio GET y POST

**$\_GET y $\_POST** son variables superglobales que forman arrays de keys y values, donde los keys son los nombres del formulario (atributo "name") y los values son los datos de entrada de los usuarios. El hecho de que sean superglobals hace que sean accesibles desde el script independientemente del ámbito.

**$\_GET** es un array de variables que se pasan al script a través de los parámetros de URL. La información que se envía es visible para todo el mundo y tiene limitada la cantidad de información que se puede enviar a 2000 caracteres. Las URLs con los datos enviados pueden guardarse en marcadores, lo que puede ser útil en ciertos casos. $\_GET sólo se emplea para el envío de información reducida y no sensible.

**$\_POST** es un array de variables que se pasan al script a través del método HTTP POST. La información que se envía no es visible para los demás ya que los nombres y variables van embebidas en el body del HTTP request. No tiene límites en la cantidad de información a enviar. Soporta datos de entrada multi-part binary a la hora se subir archivos al servidor (lo que permite subir múltiples archivos a la vez).

Nota: $\_REQUEST es un array donde se guardan todos los elementos de las que están en $\_POST, $\_GET, $\_COOKIE.

## Validación de datos del formulario en cliente

La validación de formularios es algo fundamental ya que previene posibles ataques de intrusos, además de asegurar que los datos que se reciben son realmente del tipo deseado.

Existen dos formas de validación de formularios: en el lado del cliente y en el lado del servidor. En el lado del cliente la validación suele ser mediante JavaScript, es más rápida y evita enviar más trabajo al servidor. En el lado del servidor se emplea PHP para verificar que se envían valores correctos, es más seguro pero es más lento y da un poco de trabajo al servidor. En este artículo vamos a ver algún ejemplo de esta segunda forma.

El siguiente es un sencillo formulario algo más complejo con los siguiente datos: Nombre, Contraseña, Educacion, Nacionalidad, Idiomas, Email y Sitio web:

<h2>Formulario:</h2>

<form action="mipagina.php" method="post">

Nombre:

<input type="text" name="nombre" maxlength="50"><br>

Contraseña:

<input type="password" name="password"><br>

Educacion:

<select name="educacion">

<option value="sin-estudios">Sin estudios</option>

<option value="educacion-obligatoria" selected="selected">Educación Obligatoria</option>

<option value="formacion-profesional">Formación profesional</option>

<option value="universidad">Universidad</option>

</select> <br>

Nacionalidad:

<input type="radio" name="nacionalidad" value="hispana">Hispana</input>

<input type="radio" name="nacionalidad" value="otra">Otra</input><br>

Idiomas:

<input type="checkbox" name="idiomas[]" value="español" checked="checked">Español</input>

<input type="checkbox" name="idiomas[]" value="inglés">Inglés</input>

<input type="checkbox" name="idiomas[]" value="francés">Francés</input>

<input type="checkbox" name="idiomas[]" value="aleman">Alemán</input><br>

Email:

<input type="text" name="email"><br>

Sitio Web:

<input type="text" name="sitioweb"><br>

// Botón de enviar

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

</form>

</body>

</html>

Vamos a ver los aspectos más característicos del código anterior.

El atributo **maxlength** limita el número de caracteres para una caja de texto.

* Contraseña es del tipo **password**, lo que permite ocultar los caracteres con \* cuando se escribe.
* Educación es un elemento **select**, que permite seleccionar options en una lista. El texto en el atributo value es el que se enviará a través del formulario. El atributo selected permite mostrar el seleccionado por defecto.
* Nacionalidad es del tipo **radio**, los elementos radio del mismo grupo han de tener un único atributo name y sólo se puede seleccionar uno.
* Los idiomas están en un elemento **checkbox**, que permite enviar varios datos en forma de array si se indica en el atributo name con corchetes. Se puede añadir un atributo checked como seleccionado por defecto.

Para el ejemplo anterior, la forma de recorrer los idiomas sería la siguiente:

if(!empty($\_POST[‘idiomas’])){

// Ciclo para mostrar las casillas checked checkbox.

foreach($\_POST[‘idiomas’] as $selected){

echo $selected."</br>";// Imprime resultados

}

}

Otra opción para enviar Checkbox sería nombrar de forma distinta a cada una de las opciones disponibles, pero habría que validar todos los name a la hora de enviar y recibir la información:

<input type="checkbox" name="idiomasES" value="español">Español</input>

<input type="checkbox" name="idiomasEN" value="inglés">Inglés</input>

<input type="checkbox" name="idiomasFR" value="francés">Francés</input>

<input type="checkbox" name="idiomasAL" value="aleman">Alemán</input><br>

**Atributo onsubmit**: Ejecutar un JavaScript cuando se envía un formulario, de esta manera antes de ejecutarse el action podemos validar que los campos que vamos a enviar son correctos.

Si la función llamada en onSumbit devuelve false, se para la ejecución y no se llama a la URL del atributo action.

Ejemplo:

**function** eCheck()

{

**var** filePath = document.getElementsByName("fileSelect");

**if** (filePath == '')

{

**return** false;

}

return true;

}

<**form** action="process.php" method="post" onSubmit="return eCheck();">

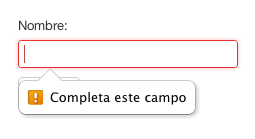
### Validaciones con HTML5

**HTML5 nos facilita mucho la validación de formularios** en el cliente, os dejo un enlace donde leer más al respecto:

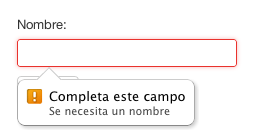
<https://cybmeta.com/validacion-de-formularios-con-html5>

**El atributo required**: Al incluir el atributo required dentro de un elemento <input>, automáticamente se hace obligatorio su llenado y al ser un atributo tipo booleano, solo se requiere su presencia sin más.

<input type="text" name="nombre" required/>

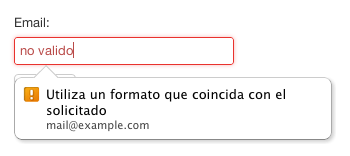


<input title="Se necesita un nombre" type="text" name="nombre" required/>



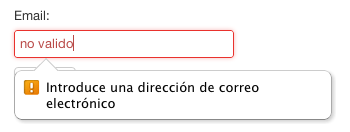
**El atributo pattern vs tipo de input**: utilizando el atributo pattern en conjunto, se logra que se verifique no solo la presencia de un valor, sino que este valor debe contener un formato, una longitud o un tipo de dato especifico. Esto último se logra definiendo un patrón con expresiones regulares.

<input type="text" pattern="^[a-zA-Z0-9.!#$%&'\*+/=?^\_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)\*$" name="email" required/>



Con el patrón anterior se valida un formato correcto de correo electrónico (mail@example.com), pero en la práctica esto resulta un tanto absurdo, ya que definiendo un tipo email en la etiqueta <input>, el navegador por si mismo hace la validación del tipo en cuestión.

<input type="email" name="email" required/>



Pasa lo mismo con el resto de los tipos de <input>: search, url, tel, email, password, date pickers, number, checkbox, radio y file. Por lo tanto el potencial del atributo pattern recae en ser más específicos en el tipo de formato y longitud que se requiere.

Por ejemplo, supongamos que se necesitamos validar un usuario de twitter:

<input type="text" name="twitter\_username" pattern="^@?(\w){1,15}$"

title="Este no parece un usuario de twitter válido">

Otro ejemplo: necesitamos un nombre de usuario que tenga en 5 y 12 caracteres.

<input type="text" name="usuario" id="usuario" pattern="^([a-z]+[0-9]{0,2}){5,12}$" required/>

Para saber más sobre expresiones regulares:

<https://www.arkaitzgarro.com/javascript/capitulo-11.html>

Con **respecto a los tipos input HTML5**, aquí tenemos algunos de los más usados:

* **type=”number”** Podemos enviar enteros, estableciendo un mínimo y un máximo como input de texto.
* **type=”range”** Podemos enviar enteros, estableciendo un mínimo y un máximo utilizando un control gráfico.
* **type=”color”** Nos permite seleccionar desde un “color picker” un color y enviarlo como hexadecimal.
* **type=”date”** Nos permite seleccionar una día concreto. En los navegadores más avanzados aparece un calendario.
* **type=”month”, type=”week”, type=”time”** Nos permiten seleccionar distintas unidades de tiempo: mes, semana o año.
* **type=”email”** Permite enviar una dirección de email con una pobre validación a nivel de cadena.
* **type=”url”** Permite enviar una url con una pobre validación a nivel de cadena.
* **type=”tel”** Para el envío de números de teléfono.
* **type=”search”** Para definir campos de búsqueda, aunque realmente se comporta como un campo de tipo texto.

## Enviando y recibiendo información

Un formulario sencillo con dos campos a rellenar y un botón para enviar en **HTML** es de la siguiente forma:

<**html**>

<**body**>

<**form** action="formpost.php" method="post">

Nombre: <**input** type="text" name="nombre"><**br**>

Email: <**input** type="text" name="email"><**br**>

<**input** type="submit" value="Enviar">

</**form**>

</**body**>

</**html**>

Cuando un usuario rellena los datos y hace click en enviar, los datos del formulario se envían para ser procesados al archivo *formpost.php*. Los datos se envían a través de HTTP con el método **HTTP Post**.

Una vez los datos están en *formpost.php* pueden mostrarse, guardarse en una base de datos o lo que se necesite. En *formpost.php*vamos a mostrarlos con echo:

<**html**>

<**body**>

Hola <?php echo $\_POST["nombre"]; ?><**br**>

Tu email es: <?php echo $\_POST["email"]; ?>

</**body**>

</**html**>

Lo mismo se puede conseguir con el método **HTTP GET**:

<**html**>

<**body**>

<**form** action="formget.php" method="get">

Nombre: <**input** type="text" name="nombre"><**br**>

Email: <**input** type="text" name="email"><**br**>

<**input** type="submit" value="Enviar">

</**form**>

</**body**>

</**html**>

Ahora el archivo donde recibimos los datos es *formget.php*:

<**html**>

<**body**>

Hola <?php echo $\_GET["nombre"]; ?><**br**>

Tu email es: <?php echo $\_GET["email"]; ?>

</**body**>

</**html**>

Estos han sido dos ejemplos muy sencillos para ver el **funcionamiento básico de un formulario**. Siempre que se crean formularios hay que tener muy en cuenta la **seguridad**, y para ello se utiliza la validación de formularios.

Las *superglobals* $\_GET y $\_POST sirven para coleccionar datos de formularios, y vamos a ver las diferencias a continuación.

Cuando queremos enviar checkbox la recepción se complica un poco más. Por ejemplo, para el formulario donde enviamos idiomas, la forma de recibir el array de idiomas sería la siguiente:

<?php

$idiomas = $\_GET['idiomas']; // GET o POST

if(empty($idiomas))

{

echo("No has seleccionado ningún idioma");

}

else

{

$n = count($idiomas);

for($i=0; $i < $n; $i++)

{

echo($idiomas[$i] . " ");

}

}

?>

## Validación de datos en el servidor (PHP)

Pese a que hayamos validado el envío de los datos enviados en el cliente, debemos hacerlos nuevamente en el servidor por si un usuario se ha saltado la validación inicial.

Para validar los datos en el servidor mediante PHP debemos recogerlos mediante el array $\_GET o $\_POST.

Os dejo algunos consejos y validaciones a realizar según el tipo de datos que recibas:

**Validaciones básicas** mediante comparaciones <>’’ , ==’’ , etc…

Ejemplos validaciones básicas: https://uniwebsidad.com/libros/javascript/capitulo-7/validacion

Funciones de PHP **isset(), is\_null() e empty()**

isset(): Determina si una variable está definida y no es NULL. Hay que usar isset() cuando necesitemos averiguar si una variable está definida en el script que estamos ejecutando, es decir, si existe.

Ejemplo: isset($\_GET[‘nombre’])

is\_null(): Para comprobar si una variable es NULL. La función is\_null() se usa cuando queremos saber si una variable es igual a NULL.

empty(): Para saber si una variable está vacía.La función empty() hay que usarla cuando queramos averiguar si una variable está vacía o no, es decir, si tiene contenido o no.

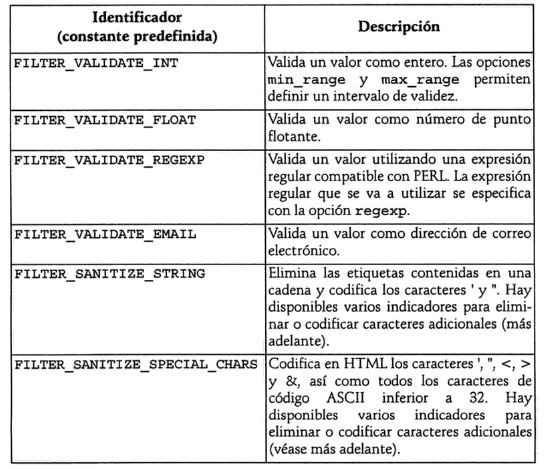
Función **is\_numeric()** se usa para verificar si una variable es numérica o no.

Mediante la orden **$\_SERVER['REQUEST\_METHOD']** podemos saber si PHP recibe los datos mediante GET o POST, en caso de no recibir los campos en el array que esperamos podemos devolver un error:

$\_SERVER['REQUEST\_METHOD'] == 'POST'

Con la función **filter\_var** podemos validar muchos campos por defecto, como el email, url, mac, ips…

Algunos ejemplos de filtros serían:



Documentación: <https://www.php.net/manual/es/filter.filters.validate.php>

A la hora de recoger cualquier dato, deberíamos aplicarle una función común para eliminar espacios en blanco o caracteres especiales:

function filtrado($datos){

$datos = trim($datos); // Elimina espacios antes y después de los datos

$datos = stripslashes($datos); // Elimina backslashes \

$datos = htmlspecialchars($datos); //Traduce caracteres especiales en entidades HTML

return $datos;

}

La función filtrado debe aplicarse a todos los datos cuando se van a recibir, por ejemplo:

$nombre = filtrado($\_POST["nombre"]);

## Subir archivos con php

Lo más laborioso a la hora de hacer un formulario que permita la subida de archivos es manejarlos y organizarlos una vez están en el servidor. Inicialmente los archivos se suben a un directorio temporal y luego se relocalizan.

Configuración en PHP para la subida de archivos:

* Debe estar **On la variable file\_uploads en php.in**i.
* En php.ini también se encuentra la variable **upload\_tmp\_dir**, que indica el directorio donde se dirigirán los archivos cuando se suban. El tamaño máximo viene indicado en el mismo archivo de configuración por **upload\_max\_filesize**. Para ver estos valores se puede utilizar ini\_get('upload\_tmp\_dir') e ini\_get('upload\_max\_filesize'). upload\_tmp\_dir sólo es manipulable desde el php.ini o httpf.conf, upload\_max\_filesize puede también desde .htaccess o un user.ini (tienen modo PHP\_INI\_SYSTEM y PHP\_INI\_PERDIRrespectivamente).
* Es necesario que los **permisos** tanto del **directorio temporal** como del directorio final sean de **escritura**.
* Para que un formulario tenga la capacidad de aceptar archivos se añade el atributo **enctype="multipart/form-data" al elemento form**.

El input necesario para subir archivos es el type file:

<input type="file" name="imagen" />

Cuando este campo se envíe al servidor, se recibiran los datos en el array multidimensional $\_FILES. Este array se crea con el key que se indique en el input del formulario, en este caso "imagen".

Los campos disponibles en el array multidimensional asociativo $\_FILES son:

* $\_FILES["imagen"]["name"] → Guarda el nombre original del archivo del cliente.
* $\_FILES["imagen"]["type"] → Guarda el MIME type del archivo.
* $\_FILES["imagen"]["size"] → Guarda el tamaño del archivo en bytes.
* $\_FILES["imagen"]["tmp\_name"] → Guarda el nombre del archivo temporal.
* $\_FILES["imagen"]["error"] → Guarda cualquier código de error que pueda provocar la subida del archivo.

El array $\_FILES['imagen']['error'] especifica por qué no se ha podido subir el archivo, lo que permite especificar un mensaje de vuelta para cada tipo de error. Devuelve un integer con el número de error:

|  |  |  |
| --- | --- | --- |
| **Error** | **Valor** | **Significado** |
| UPLOAD\_ERR\_OK | 0 | No hay errores |
| UPLOAD\_ERR\_INI\_SIZE | 1 | Supera el tamaño máximo indicado en php.ini |
| UPLOAD\_ERR\_FORM\_SIZE | 2 | Supera el tamaño máximo indicado en MAX\_FILE\_SIZE de html |
| UPLOAD\_ERR\_PARTIAL | 3 | Sólo se ha subido el archivo parcialmente |
| UPLOAD\_ERR\_NO\_FILE | 4 | No se ha subido ningún archivo |
| UPLOAD\_ERR\_NO\_TMP\_DIR | 6 | Falta la carpeta temporal |
| UPLOAD\_ERR\_CANT\_WRITE | 7 | No se puede escribir en el directorio especificado |
| UPLOAD\_ERR\_EXTENSION | 8 | Una extensión de PHP ha detenido la subida |

Para completar la operación también deberemos hacer uso de las siguientes funciones:

move\_uploaded\_file(): mueve un archivo subido del directorio temporal al directorio que se indique.

pathinfo(): Devuelve información acerca de la ruta de un fichero

$partes\_ruta = pathinfo('/www/htdocs/inc/lib.inc.php');

echo $partes\_ruta['dirname'];

echo $partes\_ruta['basename'];

echo $partes\_ruta['extension'];

echo $partes\_ruta['filename'];

Ejemplo de código PHP correspondiente a subida del archivo:

$directorioSubida = "imagenes/"; → directorio donde se guardará la imagen

$max\_file\_size = "51200"; → tamaño máximo en bytes, en este ejemplo 50 kb

$extensionesValidas = array("jpg", "png", "gif");

if(isset($\_POST["submit"]) && isset($\_FILES['imagen'])){

$errores = array();

$nombreArchivo = $\_FILES['imagen']['name'];

$filesize = $\_FILES['imagen']['size'];

$directorioTemp = $\_FILES['imagen']['tmp\_name'];

$tipoArchivo = $\_FILES['imagen']['type'];

$arrayArchivo = pathinfo($nombreArchivo);

$extension = $arrayArchivo['extension'];

// Comprobamos la extensión del archivo

if(!in\_array($extension, $extensionesValidas)){

$errores[] = "La extensión del archivo no es válida o no se ha subido ningún archivo";

}

// Comprobamos el tamaño máximo del archivo

if($filesize > $max\_file\_size){

$errores[] = "La imagen debe de tener un tamaño inferior a 50 kb";

}

// Comprobamos y renombramos el nombre del archivo para poder subir archivos con mismo nombre y no machacarlos

$nombreArchivo = $arrayArchivo['filename'];

$nombreArchivo = preg\_replace("/[^A-Z0-9.\_-]/i", "\_", $nombreArchivo);

$nombreArchivo = $nombreArchivo . rand(1, 100);

// movemos el archivo al destino final si no hay errores

if(empty($errores)){

$nombreCompleto = $directorioSubida.$nombreArchivo.".".$extension;

if(move\_uploaded\_file($directorioTemp, $nombreCompleto))

echo "El archivo se ha subido correctamente";

else echo “error”;

}

}

### Subida múltiple de archivos

Una subida múltiple de archivos en PHP en la práctica se hace normalmente con ayuda de JavaScript.

En el formulario HTML lo único que cambia es que se añade un atributo multiple y que en el atributo name del input file se añade corchetes para indicar que es un array:

<input type="file" **name="imagenes[]" multiple="multiple"** />

En cuanto al backend, omitiendo validaciones esta puede ser una forma de hacerlo:

$directorioSubida = "imagenes/";

$max\_file\_size = "51200";

if(isset($\_POST["submit"]) && isset($\_FILES['imagenes'])){

$nombres = $\_FILES['imagenes']['name'];

$temporales = $\_FILES['imagenes']['tmp\_name'];

$tipos = $\_FILES['imagenes']['type'];

$errores = $\_FILES['imagenes']['error'];

// Iteramos sobre los arrays creados

for ($i = 0; $i < count($temporales); $i++){

if(move\_uploaded\_file($temporales[$i], $directorioSubida.$nombres[$i])){

echo "Se ha subido {$nombres[$i]} correctamente <br>";

} else {

echo "Ha habido algún error al subir algún archivo";

}

}

}

La subida de archivos se puede hacer más atractiva con el uso de librerías de javascript, como por ejemplo: <https://www.dropzonejs.com/>