



Módulo

5

Programación Web. Formularios

Autor del documento:

Centro de Apoyo Tecnológico a Emprendedores

Datos de contacto:

E-Mail: bilib@bilib.es

Página Web: www.bilib.es

Teléfono: 967 555 311

Licencia del documento:

Copyright © 2017, Centro de Apoyo Tecnológico a Emprendedores

Publicado bajo licencia Creative Commons By - Sa

Usted es libre de:

- Copiar, distribuir y comunicar públicamente la obra.
- Hacer obras derivadas
- Bajo las condiciones siguientes:

Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).

Compartir bajo la misma licencia. Si transforma o modifica esta obra para crear una obra derivada, sólo puede distribuir la obra resultante bajo la misma licencia, una similar o una compatible.

Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.

Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor.

Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Los logos y marcas anunciados o referidos en este documento son propiedad de sus respectivos dueños, todos o algunos derechos reservados dependiendo de su Licencia.

1. Índice

1. Índice.....	2
2. Objetivos.	3
3. Formularios	4
3.1 Controles en formularios	4
3.2 Recogida de datos	13
3.3 Comprobación de datos	31
4. Glosario	40
5. Bibliografía	45
6. Anexos	46
7. Cuestionario de autoevaluación.	47

2. Objetivos.

Con la realización de este módulo formativo, el alumnado será capaz de:

- **Conocer** cuáles son los requisitos necesarios para configurar controles que devuelvan información.
- **Diferenciar** entre los distintos atributos posibles en función del tipo, modo y nivel de seguridad que se desee establecer para la obtención de información.
- **Conocer** el modo correcto para la configuración de “botones” en los formularios.
- **Analizar** el tipo de información que se obtiene en función del control configurado.
- **Orientar** al alumnado de los posibles errores que pueden aparecer, teniendo en cuenta las peculiaridades de PHP.
- **Dotar** al alumnado de pautas y estrategias para prever y corregir los posibles fallos antes de que estos aparezcan o una vez han aparecido.

3. Formularios

3.1 Controles en formularios

Explicación previa

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

El siguiente ejemplo muestra un formulario válido (ejemplo.html):

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

Nombre:

El programa que recibe los datos los guarda automáticamente en la matriz `$_REQUEST`. Mediante la orden `print_r($_REQUEST)` se puede mostrar el contenido de la matriz `$_REQUEST`.

El siguiente ejemplo muestra lo que escribiría el programa PHP (ejemplo.php) si recibiera la información del formulario anterior (ejemplo.html).

```
<?php
print "<pre>";
print_r($_REQUEST);
print "</pre>\n";
?>
```

```
Array
(
    [nombre] => Pepito Conejo
)
```

El nombre del elemento de la matriz `$_REQUEST` coincide con el nombre del control (atributo `name`) en el formulario (excepto en el control de tipo imagen).

En los ejemplos que se encuentran en esta página, cada ejemplo muestra el código fuente de la página HTML, el aspecto que tiene el control y el contenido de la matriz `$_REQUEST` cuando se recibe la información.

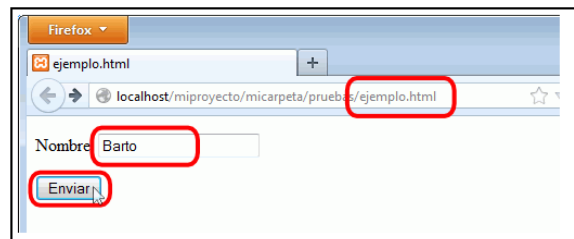
Atributo method

El atributo method de la etiqueta <form> permite elegir si la información de los controles se incluye en la llamada a la página (method="get") o se proporciona posteriormente (method="post"). Si el atributo no está definido, la información se incluye. Desde el punto de vista de la seguridad ambos métodos son equivalentes.

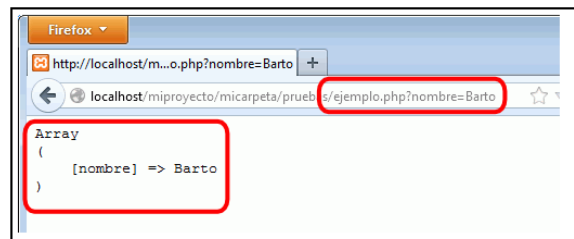
La diferencia es que con el valor get se pueden ver en la barra de dirección los nombres de los controles y los valores introducidos por el usuario, mientras que con el valor post no, pero los datos recibidos son idénticos.

• Formulario con get

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

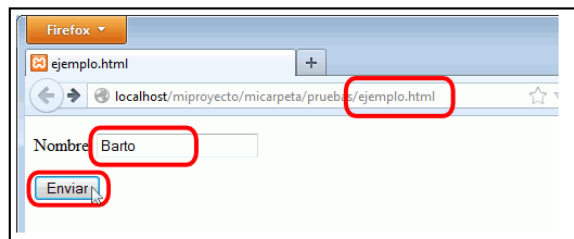


```
print "<pre>";
print_r($_REQUEST);
print "</pre>";
```

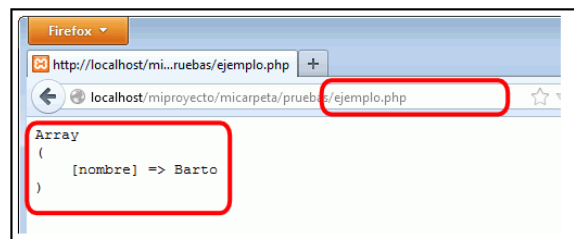


• Formulario con post

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

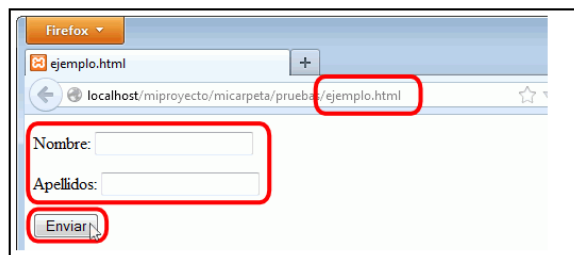


```
print "<pre>";
print_r($_REQUEST);
print "</pre>";
```

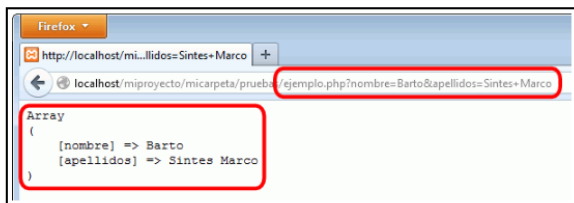


En caso de que haya varios controles que envíen información en un formulario con get, los nombres y valores aparecen en la barra de dirección separados por el carácter ampersand (&), como nombre1=valor1&nombre2=valor2&...

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



```
print "<pre>";
print_r($_REQUEST);
print "</pre>";
```



Botón Enviar (input submit, button)

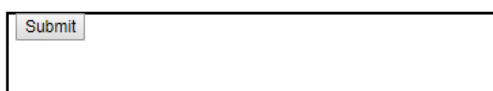
Este control se puede crear con la etiqueta `<input>` o con la etiqueta `<button>`. En ambos casos, este control se envía siempre que esté definido el atributo `name` y el valor enviado es el valor del atributo `value` o el contenido de la etiqueta.

Código fuente

Control

\$_REQUEST

```
<input type="submit" value="Submit" />
```



```
Array
(
)
```

```
<input type="submit" value="Enviar"
name="boton2" />
```



```
Array
(
    [boton2] => Enviar
)
```

Código fuente

Control

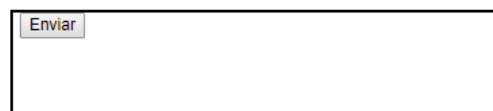
\$_REQUEST

```
<button type="submit">Submit</button>
```



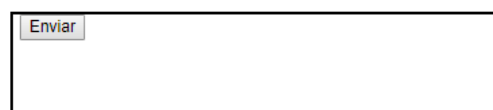
```
Array
(
)
```

```
<button type="submit"
name="boton3">Enviar</button>
```



```
Array
(
    [boton3] => Enviar
)
```

```
<button type="submit" name="boton4"
value="enviado">Enviar</button>
```

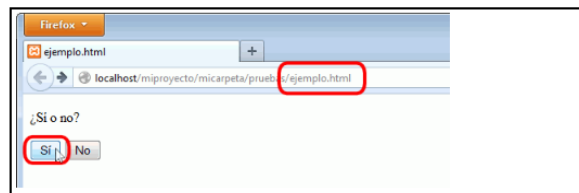


```
Array
(
    [boton4] => enviado
)
```

Normalmente no se suele dar el atributo name al botón enviar ya que la información se envía cuando se hace clic en el botón. Pero en algunos casos sí que puede ser conveniente. Por ejemplo cuando el formulario contiene varios botones y queremos saber cuál se ha pulsado, como en el ejemplo siguiente, en el que los atributos name son iguales, pero los atributos value son distintos:

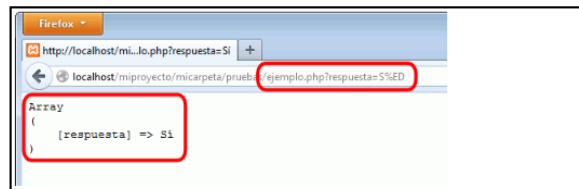
ejemplo.html

```
<form action="ejemplo.php" method="get">
<p>¿Si o no?</p>
<p><input type="submit" name="respuesta" value="Si" /></p>
<p><input type="submit" name="respuesta" value="No" /></p>
</form>
```



ejemplo.php

```
print "<pre>";
print_r($_REQUEST);
print "</pre>";
```



Caja de texto, caja de contraseña y área de texto (input text, input password, 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] =>)
<code><textarea rows="4" cols="20" name="areadetexto2">Cualquier cosa</textarea></code>	<div><div>Cualquier cosa</div></div>	Array ([areadetexto2] => Cualquier cosa)

Casilla de verificación (input 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 o el valor del atributo value si éste está definido.

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)

Botón radio (input 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 o el valor del atributo value si éste está definido.

Código fuente

Control

\$_REQUEST

<pre><input type="radio" name="radio1" /> <input type="radio" name="radio1" /></pre>	<input type="radio"/> <input type="radio"/> (sin marcar)	Array ()
<pre><input type="radio" name="radio2" /> <input type="radio" name="radio2" /></pre>	<input checked="" type="radio"/> <input type="radio"/> (marcado uno)	Array ([radio2] => on)
<pre><input type="radio" name="radio3" /> <input type="radio" name="radio3" /></pre>	<input type="radio"/> <input checked="" type="radio"/> (marcado otro)	Array ([radio3] => on)
<pre><input type="radio" name="radio4" value="Uno" /> <input type="radio" name="radio4" value="Dos" /></pre>	<input checked="" type="radio"/> <input type="radio"/> (marcado uno)	Array ([radio4] => Uno)
<pre><input type="radio" name="radio5" value="Uno" /> <input type="radio" name="radio5" value="Dos" /></pre>	<input type="radio"/> <input checked="" type="radio"/> (marcado otro)	Array ([radio5] => Dos)

Menú (select)

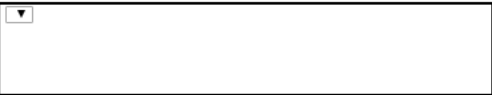

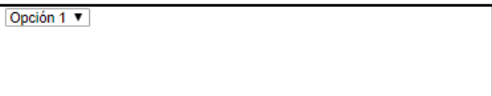

Este control envía siempre la opción elegida. El valor enviado es el contenido de la etiqueta option elegida si la opción elegida no tiene definido el atributo value o el valor del atributo value si éste está definido.

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.

Código fuente

Control

\$_REQUEST

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

Control oculto (input hidden)

Este control se envía siempre y el valor enviado es el valor del atributo value.

Código fuente

Control

\$_REQUEST



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

Imagen (input image)

El control de tipo imagen inserta una imagen que funciona como un botón (aunque ni Firefox ni Internet Explorer le da relieve como a los botones). Al hacer clic en un punto de la imagen es como si se hubiera pulsado a un botón submit y se envían las coordenadas del punto en el que se ha hecho clic (junto con los valores de los otros controles del formulario).

El origen de las coordenadas es el extremo superior izquierdo de la imagen. El valor X aumenta a medida que nos desplazamos a la derecha y el valor Y aumenta a medida que nos desplazamos hacia abajo.

Código fuente

```
<input type="image" name="gnu" alt="Logotipo GNU"
src="gnu.jpg" />
```

Control



\$_REQUEST

```
Array
(
    [gnu_x] => 89
    [gnu_y] => 111
)
```

Si se define el atributo value, el formulario debe enviar tanto las coordenadas como el nombre del control con el valor del atributo value. Actualmente (octubre de 2016), Google Chrome sí que lo hace, pero Firefox e Internet Explorer no lo hacen.

Archivo (input file)

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

En un formulario "normal", este control se envía siempre y el valor enviado es el nombre del archivo elegido.

Código fuente

```
<input type="file" name="archivo1" />
```

Control

Seleccionar archivo Ningún archivo seleccionado

\$_REQUEST

```
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 con el método POST. La información se almacena entonces en la matriz \$_FILES (pero no en la variable \$_REQUEST).

Código fuente

```
<form enctype="multipart/form-data"
  action="ejemplo.php" method="post" >
<input type="file" name="archivo2" />
```

Control

Seleccionar archivo | Ningún archivo seleccionado

\$_REQUEST

```
Array
(
    [archivo2] => Array
        (
            [name] => loquesea.txt
            [type] => text/plain
            [tmp_name] => C:\ejemplos\loquesea.txt
            [error] => 0
            [size] => 27890
        )
)
```

Antes de utilizar este control, hay que configurar en el archivo php.ini el tamaño máximo de los archivos que se pueden enviar (mediante la directiva post_max_size).

3.2 Recogida de datos

Matriz \$_REQUEST

Cuando se envía un formulario, PHP almacena la información recibida en una matriz llamada [\\$_REQUEST](#). El número de valores recibidos y los valores recibidos dependen tanto del formulario como de la acción del usuario.

Cualquier control se envía solamente si está establecido su atributo name. El atributo name del control puede contener cualquier carácter (números, acentos, guiones, etc.), pero si contiene espacios, los 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 (en su caso).

El siguiente ejemplo muestra un ejemplo de formulario:

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

Nombre:

Mientras se está programando, para comprobar que el fichero php está recibiendo la información enviada por el control, lo más fácil es utilizar la función [print_r\(\\$matriz\)](#) para mostrar el contenido de la matriz \$_REQUEST. Una vez se ha comprobado que la información llega correctamente, la línea se debe comentar o eliminar.

El siguiente ejemplo muestra lo que escribiría el programa PHP si recibiera la información del formulario anterior.

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

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

Referencia a `$_REQUEST` dentro y fuera de cadenas

Al hacer referencia a los elementos de la matriz `$_REQUEST`, hay que tener en cuenta si la referencia se encuentra dentro de una cadena o fuera de ella.

- Si se hace referencia dentro de una cadena, el índice se debe escribir sin comillas. Si se escribe cualquier tipo de comillas (simples o dobles) se produce un error.

✓	<pre><?php print "<p>Su nombre es \$_REQUEST[nombre]</p>\n"; ?></pre>	Su nombre es Pepito Conejo
✗	<pre><?php print "<p>Su nombre es \$_REQUEST['nombre']</p>\n"; ?></pre>	Parse error: syntax error, unexpected "" (T_ENCAPSED_AND_WHITESPACE), expecting identifier (T_STRING) or variable (T_VARIABLE) or number (T_NUM_STRING) in ejemplo.php on line 2
✗	<pre><?php print "<p>Su nombre es \$_REQUEST['nombre']</p>\n"; ?></pre>	Parse error: syntax error, unexpected " , expecting identifier (T_STRING) or variable (T_VARIABLE) or number (T_NUM_STRING) in ejemplo.php on line 2

- Si se hace referencia fuera de una cadena, el índice se debe escribir con comillas dobles o simples. Si se escribe sin comillas, se produce un aviso.



```
<?php
print "<p>Su nombre es " . $_REQUEST['nombre'] . "</p>\n";
?>
```

Su nombre es Pepito Conejo



```
<?php
print "<p>Su nombre es " . $_REQUEST["nombre"] . "</p>\n";
?>
```

Su nombre es Pepito Conejo



```
<?php
print "<p>Su nombre es " . $_REQUEST[nombre] . "</p>\n";
?>
```

Notice: Use of undefined constant nombre - assumed 'nombre' in ejemplo.php on line 2
Su nombre es Pepito Conejo

En ningún caso se pueden utilizar los caracteres especiales \" o \', ni dentro ni fuera de las cadenas.



```
<?php
print "<p>Su nombre es $_REQUEST[\"nombre\"]</p>\n";
?>
```

Parse error: syntax error, unexpected " (T_ENCAPSED_AND_WHITESPACE), expecting identifier (T_STRING) or variable (T_VARIABLE) or number (T_NUM_STRING) in ejemplo.php on line 2



```
<?php
print "<p>Su nombre es " . $_REQUEST[\'nombre\'] . "</p>\n";
?>
```

Parse error: syntax error, unexpected " (T_ENCAPSED_AND_WHITESPACE), expecting identifier (T_STRING) or variable (T_VARIABLE) or number (T_NUM_STRING) in ejemplo.php on line 2

Comprobación de existencia

Un programa PHP no debe suponer que los controles le van a llegar siempre porque cuando eso no ocurra, el programa no funcionará correctamente.

- **Controles vacíos**

La primera situación que los programas deben tener en cuenta es que los controles no contengan ningún valor.

En el programa de ejemplo del principio de esta lección, si el usuario ha utilizado el formulario y ha escrito un dato, el programa PHP recibe el control y funciona correctamente:

ejemplo.html

ejemplo.php

http:// ... /ejemplo.php?
nombre=Pepito+Conejo

Nombre:

```
<?php
print "<pre>"; print_r($_REQUEST); print "</pre>\n";
print "<p>Su nombre es $_REQUEST[nombre]</p>\n";
?>
```

```
Array
(
    [nombre] => Pepito Conejo
)
Su nombre es Pepito Conejo
```

Pero si el usuario no escribe nada en el formulario, aunque el programa funcione correctamente, la respuesta del programa puede confundir al usuario:

ejemplo.html

ejemplo.php

http:// ... /ejemplo.php?nombre=

Nombre:

```
<?php
print "<pre>"; print_r($_REQUEST); print "</pre>\n";
print "<p>Su nombre es $_REQUEST[nombre]</p>\n";
?>
```

```
Array
(
    [nombre] =>
)
Su nombre es
```

Este problema se puede resolver incluyendo una estructura if ... else ... que considere la posibilidad de que no se haya escrito nada en el formulario:

ejemplo.html

ejemplo.php

http:// ... /ejemplo.php?nombre=

Nombre:

```
<?php
print "<pre>"; print_r($_REQUEST); print "</pre>\n";
if ($_REQUEST["nombre"] == "") {
    print "<p>No ha escrito ningún nombre</p>";
} else {
    print "<p>Su nombre es $_REQUEST[nombre]</p>\n";
}
?>
```

```
Array
(
    [nombre] =>
)
No ha escrito ningún nombre
```

- **Controles inexistentes: isset()**

Un problema más grave es que el programa suponga que existe un control que en realidad no le ha llegado.

Eso puede ocurrir, por ejemplo, en el caso de las casillas de verificación y los botones radio, ya que los formularios envían el control solamente cuando se han marcado.

```
<form action="ejemplo.php">
  <p>Deseo recibir información: <input type="checkbox" name="acepto" /></p>
  <p><input type="submit" value="Enviar" /></p>
</form>
```

Deseo recibir información: ☐

Enviar

Si en el formulario anterior el usuario marca la casilla, el siguiente programa funciona perfectamente:

ejemplo.html

ejemplo.php

http:// ... /ejemplo.php?acepto=on

Deseo recibir información: ☒

Enviar

```
<?php
print "<pre>"; print_r($_REQUEST); print "</pre>\n";

if ($_REQUEST["acepto"] == "on") {
    print "<p>Deseo recibir información</p>\n";
} else {
    print "<p>No desea recibir información</p>\n";
}
?>
```

```
Array
(
    [acepto] => on
)
Deseo recibir información
```

Pero si el usuario no marca la casilla, la matriz `$_REQUEST` no contiene el dato y el programa genera un aviso por hacer referencia a un índice no definido, aunque se haya incluido la estructura `if ... else ...` :

ejemplo.html

ejemplo.php

http:// ... /ejemplo.php

Deseo recibir información: ☐

Enviar

```
<?php
print "<pre>"; print_r($_REQUEST); print "</pre>\n";

if ($_REQUEST["acepto"] == "on") {
    print "<p>Deseo recibir información</p>\n";
} else {
    print "<p>No desea recibir información</p>\n";
}
?>
```

```
Array
(
)
Notice: Undefined index: acepto in ejemplo.php on line 4
No desea recibir información
```

Esto puede ocurrir en realidad en cualquier formulario (incluso en formularios con cajas de texto que en principio envían siempre el control) ya que el usuario puede

escribir directamente la dirección del programa PHP en el navegador y ejecutar el programa PHP sin pasar por el formulario. En ese caso, el programa no recibe ningún control y el programa genera un aviso por hacer referencia a un índice no definido:

ejemplo.html

Nombre:

ejemplo.php

```
<?php
print "<pre>"; print_r($_REQUEST); print "</pre>\n";

if ($_REQUEST["nombre"] == "") {
    print "<p>No ha escrito ningún nombre</p>";
} else {
    print "<p>Su nombre es $_REQUEST[nombre]</p>\n";
}
?>
```

http:// ... /ejemplo.php

```
Array
(
)
Notice: Undefined index: nombre in ejemplo.php on line 4
No ha escrito ningún nombre
```

Estos problemas se pueden resolver comprobando que el índice está definido antes de hacer referencia a él, utilizando la función [isset\(\\$variable\)](#), que admite como argumento una variable y devuelve **true** si existe y **false** si no existe.

ejemplo.html

Deseo recibir información: ☒

ejemplo.php

```
<?php
if (isset($_REQUEST["acepto"])) {
    print "<p>Desea recibir información</p>\n";
} else {
    print "<p>No desea recibir información</p>\n";
}
?>
```

http:// ... /ejemplo.php?acepto=on

No desea recibir información

ejemplo.html

Deseo recibir información: ☐

ejemplo.php

```
<?php
if (isset($_REQUEST["acepto"])) {
    print "<p>Desea recibir información</p>\n";
} else {
    print "<p>No desea recibir información</p>\n";
}
?>
```

http:// ... /ejemplo.php

No desea recibir información

Es conveniente efectuar siempre la verificación de existencia para prevenir los casos en que un usuario intente acceder a la página PHP sin pasar por el formulario.

Seguridad en las entradas

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:

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

El siguiente ejemplo muestra cómo puede utilizarse un formulario para inyectar en la página código JavaScript. En este caso el código genera una ventana emergente que sólo afecta al propio usuario, pero esta vía podría utilizarse para atacar al servidor.

Nombre: <input type="text" value="<body onload="alert('Hola')""/> <input type="button" value="Enviar"/>	<pre><?php print "<pre>"; print_r(\$_REQUEST); print "</pre>\n"; print "<p>Su nombre es \$_REQUEST[nombre]</p>\n"; ?></pre>	
---	--	--

Estos dos ejemplos no pondrían el riesgo el servidor, pero si en una aplicación web los datos que envía el usuario se incluyen sin precaución en una consulta SQL estaríamos hablando de una vía de ataque al servidor muy peligrosa.

- **Eliminar etiquetas: `strip_tags($cadena)`**

Una manera de resolver el problema anterior es utilizar la función [strip_tags\(\\$cadena\)](#), que devuelve la cadena sin etiquetas, como muestra el siguiente ejemplo.

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

Aunque hay que tener en cuenta que esta función elimina cualquier cosa que se interprete como una etiqueta, es decir, que empiece por "<".

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

- **Eliminar espacios en blanco iniciales y finales: trim(\$cadena)**

A veces, al rellenar un formulario, los usuarios escriben por error espacios en blanco al principio o al final de las cajas de texto y si el programa PHP no tiene en cuenta esa posibilidad se pueden obtener resultados inesperados.

En el programa de ejemplo del apartado anterior en el que se escribía una estructura if ... else ... para avisar al usuario si no escribía el nombre, si el usuario escribe únicamente espacios en blanco, el programa funciona, pero no da la respuesta adecuada ya que la cadena con espacios en blanco no es una cadena vacía.

ejemplo.html

ejemplo.php

http:// ... /ejemplo.php?nombre=

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

Este problema se puede resolver utilizando la función [trim\(\\$cadena\)](#), que elimina los espacios en blanco iniciales y finales y devuelve la cadena sin esos espacios. Modificando el ejemplo anterior, la cadena introducida queda reducida a la cadena vacía y la comprobación la detecta:

ejemplo.html

Nombre:

ejemplo.php

```
<?php
print "<pre>"; print_r($_REQUEST); print "</pre>\n";
if (trim($_REQUEST["nombre"]) == "") {
    print "<p>No ha escrito ningún nombre</p>\n";
} else {
    print "<p>Su nombre es ". trim($_REQUEST["nombre"])
    . "</p>\n";
}
?>
```

http:// ... /ejemplo.php?nombre=

```
Array
(
    [nombre] =>
)
No ha escrito ningún nombre
```

Utilización de variables

Hemos visto que para resolver los problemas comentados en los apartados anteriores, al incluir cualquier referencia a `$_REQUEST[control]` habría que:

- comprobar que el control esté definido con la función **isset()**
- eliminar las etiquetas con la función **strip_tags()**
- eliminar los espacios en blanco iniciales y finales con la función **trim()**

Una manera de hacerlo sin complicar excesivamente el programa es guardar los valores de la matriz `$_REQUEST` en variables y realizar todas las comprobaciones al definir esas variables. En el resto del código basta con utilizar la variable en vez del elemento de la matriz `$_REQUEST`. En los siguientes ejemplos, se puede ver cómo el programa no genera avisos y muestra el mensaje correspondiente al dato recibido:

ejemplo.html

Nombre:

ejemplo.php

```
<?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>\n";
} else {
    print "<p>Su nombre es $nombre</p>\n";
}
?>
```

http:// ... /ejemplo.php?nombre=

```
<pre>Array
(
    [nombre] =>
)
</pre>
<p>No ha escrito ningún nombre</p>
```

ejemplo.html

Nombre:

ejemplo.php

```
<?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>\n";
} else {
    print "<p>Su nombre es $nombre</p>\n";
}
?>
```

http:// ...
/ejemplo.php?nombre=Pe
+Conejo<%2Fstrong>

```
<pre>Array
(
    [nombre] => <strong>Pepito Conejo</strong>
)
</pre>
<p>Su nombre es Pepito Conejo</p>
```

La asignación de la variable se puede realizar en una sola instrucción, utilizando la notación abreviada: (condición) ? verdadero : falso; (que se explica en la lección de [estructuras de control](#)):

ejemplo.html

Nombre:

ejemplo.php

```
<?php
print "<pre>"; print_r($_REQUEST); print "</pre>\n";

$nombre = (isset($_REQUEST["nombre"]))
? trim(strip_tags($_REQUEST["nombre"]))
: "";

if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>\n";
} else {
    print "<p>Su nombre es $nombre</p>\n";
}
?>
```

http:// ...
/ejemplo.php?nombre=Pe
+Conejo<%2Fstrong>

```
<pre>Array
(
    [nombre] => <strong>Pepito Conejo</strong>
)
</pre>
<p>Su nombre es Pepito Conejo</p>
```

Salida de datos

Si, como en los ejemplos anteriores, los datos recogidos se van a incorporar luego a una página web, hay que tener cuidado en algunos casos especiales. Se comentan a continuación dos de ellos y después se comenta una solución general para este tipo de caracteres.

- **El carácter & (ampersand)**

Si el usuario escribe en una entrada el carácter & (ampersand, entidad de carácter &) y esa cadena se escribe en una página, la página se verá correctamente en el navegador, pero la página no será válida (el xhtml será inválido).

La página es inválida porque el carácter & indica el comienzo de una [entidad de carácter](#), así que no el código fuente no puede haber un & sin nada a continuación. Si se quiere mostrar un carácter & en el navegador, en el código fuente debe escribirse la entidad de carácter &.



Nombre:

Enviar

```
<?php
$nombre = (isset($_REQUEST["nombre"]))
? trim(strip_tags($_REQUEST["nombre"]))
: "";

if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>\n";
} else {
    print "<p>Su nombre es $nombre</p>\n";
}
?>
```

<p>Su nombre es Pepito & Company</p>

La solución es sustituir el carácter & por su entidad de carácter correspondiente (&). Eso se puede hacer con la función [str_replace\(\)](#)



Nombre:

Enviar

```
<?php
$nombre = (isset($_REQUEST["nombre"]))
? trim(strip_tags($_REQUEST["nombre"]))
: "";

$nombre = str_replace('&', '&amp;', $nombre);

if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>\n";
} else {
    print "<p>Su nombre es $nombre</p>\n";
}
?>
```

<p>Su nombre es Pepito & Company</p>

- El carácter " (comillas)

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.



Nombre:

Enviar

```
<?php
$nombre = (isset($_REQUEST["nombre"]))
? trim(strip_tags($_REQUEST["nombre"]))
: "";

if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>\n";
} else {
    print "<p>Corrige: <input type='text'
value='\"$nombre\"' /></p>\n";
}
?>
```

Corrige:

El problema es que el código fuente contiene comillas dentro de comillas y el navegador no puede interpretar correctamente el código (en este caso, el valor del atributo value es solamente "Me llamo":

```
<p>Corrige: <input type="text" value="Me llamo "Pepe" /></p>
```

Como en el caso anterior, la solución es sustituir el carácter " por su entidad de carácter correspondiente ("). Eso se puede hacer con la función `str_replace()`



Nombre:

```
<?php
$nombre = (isset($_REQUEST["nombre"]))
    ? trim(strip_tags($_REQUEST["nombre"]))
    : "";
$nombre = str_replace("'", '&quot;', $nombre);

if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>\n";
} else {
    print "<p>Corrija: <input type='text'
value='\"$nombre\"' /></p>\n";
}
?>
```

Corrija:

- **Solución general: `htmlspecialchars()`**

Además de los caracteres ampersand (&) y comillas ("), también podrían dar problemas los caracteres comillas simples (') o las desigualdades (< y >). Se podría sustituir cada uno de ellos como se ha hecho en los ejemplos anteriores, pero la función [htmlspecialchars\(\)](#) realiza todas las sustituciones de una sola vez.

Nombre:

```
<?php
$nombre = (isset($_REQUEST["nombre"]))
    ? htmlspecialchars(trim(strip_tags($_REQUEST["nombre"])),
    ENT_QUOTES, "UTF-8")
    : "";

if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>\n";
} else {
    print "<p>Corrija: <input type='text'
value='\"$nombre\"' /></p>\n";
}
?>
```

Corrija:

El ejemplo anterior tiene el inconveniente de que la primera función que se aplica es [strip_tags\(\)](#) por lo que se eliminará todo lo que esté entre marcas (<>).

Nombre:

```
<?php
$nombre = (isset($_REQUEST["nombre"]))
    ? htmlspecialchars(trim(strip_tags($_REQUEST["nombre"])),
    ENT_QUOTES, "UTF-8")
    : "";

if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>\n";
} else {
    print "<p>Corrija: <input type='text'
value='\"$nombre\"' /></p>\n";
}
?>
```

Corrija:

Si no queremos quitar etiquetas, pero no queremos que se consideren como etiquetas sino como texto, habría que aplicar primero la función [htmlspecialchars\(\)](#), como en el ejemplo siguiente.

<p>Nombre: <input type="text" value="Me llamo Pepe</>"/></p> <p><input type="button" value="Enviar"/></p>	<pre><?php \$nombre = (isset(\$_REQUEST["nombre"])) ? strip_tags(trim(htmlspecialchars(\$_REQUEST["nombre"], ENT_QUOTES, "UTF-8"))) : ""; if (\$nombre == "") { print "<p>No ha escrito ningún nombre</p>\n"; } else { print "<p>Corrija: <input type=\"text\" value=\"\\$nombre\" /></p>\n"; } ?></pre>	<p>Corrija: <input type="text" value="Me llamo Pepe</>"/></p>
---	---	---

En el ejemplo anterior la función [strip_tags\(\)](#) no hace nada puesto que las marcas < y > se han cambiado antes por las entidades de carácter < y > así que se podría simplificar el programa:

<p>Nombre: <input type="text" value="Me llamo Pepe</>"/></p> <p><input type="button" value="Enviar"/></p>	<pre><?php \$nombre = (isset(\$_REQUEST["nombre"])) ? trim(htmlspecialchars(\$_REQUEST["nombre"], ENT_QUOTES, "UTF-8")) : ""; if (\$nombre == "") { print "<p>No ha escrito ningún nombre</p>\n"; } else { print "<p>Corrija: <input type=\"text\" value=\"\\$nombre\" /></p>\n"; } ?></pre>	<p>Corrija: <input type="text" value="Me llamo Pepe</>"/></p>
---	---	---

Funciones de recogida de datos

La forma más cómoda de tener en cuenta todos los aspectos comentados en los puntos anteriores es definir una función recoge():

- La función tendrá como argumento el nombre del control que se quiere recibir y devolverá el valor recibido (o una cadena vacía si el control no se ha recibido).
- Para tratar los datos recibidos se aplicarán únicamente las funciones **htmlspecialchars()** y **trim()**:

```
trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES, "UTF-8"))
```

De esta manera, si se reciben etiquetas (<>), las etiquetas no se borrarán, sino que se conservarán como texto. Si se quisieran borrar las etiquetas, habría que aplicar las funciones **strip_tags()**, **trim()**, y **htmlspecialchars()**:

```
htmlspecialchars(trim(strip_tags($_REQUEST[$var])), ENT_QUOTES, "UTF-8")
```

- Una vez definida la función, al comienzo del programa se almacenarán en variables los datos devueltos por la función .
- En el resto del programa se trabaja con las variables.

- **Recoger un dato**

La siguiente función se podrá utilizar en la mayoría de ejercicios propuestos en estas lecciones.

```
<?php
// FUNCIÓN DE RECOGIDA DE UN DATO
function recoge($var)
{
    $tmp = (isset($_REQUEST[$var]))
        ? trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES, "UTF-8"))
        : "";
    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>\n";
}
?>
```

- **Recoger una matriz de una dimensión**

La siguiente función se podrá utilizar en los ejercicios propuestos en estas lecciones (en los que se recojan matrices).

Si un formulario envía los datos en forma de matriz, como en el ejemplo siguiente, la función recoge() del punto anterior no serviría.

```
<form action="ejemplo.php">
<p>Nombre: <input type="text" name="nombre[1]" /></p>
<p>Apellido: <input type="text" name="nombre[2]" /></p>
<p><input type="submit" value="Enviar" /></p>
</form>
```

Nombre:	<input type="text" value="Pepito"/>
Apellido:	<input type="text" value="Conejo"/>
<input type="button" value="Enviar"/>	

Hay que hacer otra función que recoja y trate los datos en forma de matriz.

- La función recoge Matriz() del ejemplo siguiente tiene como argumento el nombre del control que se quiere recibir (que debe ser una matriz de una dimensión, no sirve para matrices de dos o más dimensiones) y devuelve una matriz con los valores recibidos o una matriz vacía si el control no se ha recibido.

```
<?php
// FUNCIÓN DE RECOGIDA DE UNA MATRIZ DE UNA DIMENSIÓN
function recogeMatriz($var)
{
    $tmpMatriz = [];
    if (isset($_REQUEST[$var]) && is_array($_REQUEST[$var])) {
        foreach ($_REQUEST[$var] as $indice => $valor) {
            $indiceLimpio = trim(htmlspecialchars($indice, ENT_QUOTES, "UTF-8"));
            $valorLimpio = trim(htmlspecialchars($valor, ENT_QUOTES, "UTF-8"));
            $tmpMatriz[$indiceLimpio] = $valorLimpio;
        }
    }
    return $tmpMatriz;
}

// EJEMPLO DE USO DE LA FUNCIÓN ANTERIOR
$nombre = recogeMatriz("nombre");

if ($nombre[1] == "") {
    print "<p style='color: red'>No ha escrito su nombre.</p>\n";
} else {
    print "<p>Su nombre es <strong>$nombre[1]</strong>.</p>\n";
}

if ($nombre[2] == "") {
    print "<p style='color: red'>No ha escrito su apellido.</p>\n";
} else {
    print "<p>Su apellido es <strong>$nombre[2]</strong>.</p>\n";
}
?>
```

- **Recoger un dato con valor predeterminado**

La función `recoge()` anterior se puede modificar para definir un valor predeterminado (es decir, que si el dato no existe, la función devuelve el valor enviado como segundo argumento y si no se envía el valor predeterminado la función devolvería la cadena vacía). En el ejemplo siguiente, si no se recibe el nombre, se le asigna el nombre "pobrecito hablador".

```
<?php
// FUNCIÓN DE RECOGIDA DE UN DATO CON VALOR PREDETERMINADO
function recoge($var, $var2="")
{
    $tmp = (isset($_REQUEST[$var]) && trim(strip_tags($_REQUEST[$var])) != "")
        ? trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES, "UTF-8"))
        : trim(htmlspecialchars($var2, ENT_QUOTES, "UTF-8"));
    return $tmp;
}

// EJEMPLO DE USO DE LA FUNCIÓN ANTERIOR
$nombre = recoge("nombre", "pobrecito hablador");
print "<p>Su nombre es $nombre</p>\n";
?>
```

- **Recoger y recortar los datos**

En caso de que por algún motivo se quiera recortar el tamaño de los datos recibidos a un tamaño determinado se puede crear una función `recorta()` que recorte la longitud de los campos y llamarla desde la función `recoge()`:

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

$recorta = [
    "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[$var]))
        ? trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES, "UTF-8"))
        : "";
    $tmp = recorta($var, $tmp);
    return $tmp;
}

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

3.3 Comprobación de datos

Antes de utilizar en un programa los datos recibidos a través de un formulario, es necesario comprobar si el dato recibido corresponde a lo esperado. Algunas comprobaciones se pueden hacer con comparaciones (si no es vacío, si es un valor comprendido entre unos valores determinado, etc.), pero antes de hacer esas comparaciones hay que comprobar que es del tipo esperado (número entero o decimal, texto, etc.) para procesarlo sin error.

En esta lección se comentan una serie de familias de funciones que permiten comprobar la existencia, el tipo o el contenido de un dato:

1. funciones **is_**
2. funciones **ctype_**
3. funciones **filter_**
4. funciones **_exists**

Pero antes se comenta un caso especial, la comprobación de números, para la que se aconseja utilizar las funciones `is_numeric()` y `ctype_digit()`.

Comprobación de números con `is_numeric()` y `ctype_digit()`

Cuando se quiere comprobar que un dato recibido a través de un formulario es un número, en el caso de las funciones **is_** o **ctype_**, sólo tiene sentido utilizar dos de ellas:

- **Comprobación de números con `is_numeric()`**

Para comprobar si el dato que se ha recibido es un número (o se puede interpretar como número) se debe utilizar la función `is_numeric($valor)`. Esta función lógica devuelve `true` si el argumento se puede interpretar como número y `false` si no lo es.

El motivo por el que se debe utilizar la función `is_numeric($valor)`, es que lo que se recibe a través de un formulario se guarda siempre como cadena (aunque sea un número). La función `is_numeric($valor)` es la única que comprueba si el argumento se puede interpretar como número (aunque el argumento sea de tipo cadena), la función `is_int($valor)` o `is_float($valor)` hacen solamente comprobaciones sobre el tipo de los datos y devolverían siempre `false`.

- **Comprobación de números enteros positivos con `ctype_digit()`**

Para comprobar si el dato que se ha recibido es un número entero positivo (sin decimales) se puede utilizar la función `ctype_digit($valor)`. Esta función lógica devuelve `true` si todos los caracteres del argumento son dígitos (0, 1, ..., 9) y `false` si no lo son.

Funciones is_

Las funciones is_ son un conjunto de funciones booleanas que devuelven **true** si el argumento es de un tipo de datos determinado y **false** si no lo son.

	Función	Tipo de datos	alias (funciones equivalentes)
existencia	<u>isset(\$valor)</u>	devuelve si el dato está definido o no	
	<u>is_null(\$valor)</u>	null	
números	<u>is_bool(\$valor)</u>	booleano	
	<u>is_numeric(\$valor)</u>	número (puede tener signo, parte decimal y estar expresado en notación decimal, exponencial o hexadecimal).	
	<u>is_int(\$valor)</u>	entero	<u>is_integer(\$valor)</u> , <u>is_long(\$valor)</u>
	<u>is_float(\$valor)</u>	float	<u>is_double(\$valor)</u> , <u>is_real(\$valor)</u>
cadenas	<u>is_string(\$valor)</u>	cadena	
otros	<u>is_scalar(\$valor)</u>	escalar (entero, float, cadena o booleano)	
	<u>is_array(\$valor)</u>	matriz	
	<u>is_callable(\$valor)</u>	función	
	<u>is_object(\$valor)</u>	object	
	<u>is_resource(\$valor)</u>	recurso	

Si un dato es demasiado grande para el tipo de variable, las funciones devolverán **false**. Por ejemplo, si se usa como argumento un entero mayor que **PHP_INT_MAX** (2147483647 en Windows), la función **is_int()** devolverá **false**.

Estas funciones son en general de poca utilidad con datos provenientes de un formulario, ya que estas funciones lo que comprueban es el tipo de los datos y la información que llega de un formulario es siempre del tipo cadena. Las dos excepciones serían:

- la función **is_numeric()**, que evalúa si el argumento se puede interpretar como número (aunque el tipo sea cadena). Por tanto esta función se puede utilizar para comprobar si un dato recibido es un número.
- la función **isset()**, que evalúa si el argumento está o no definido, independientemente de su tipo.

El ejemplo siguiente muestra el uso de la función **is_numeric()**.

```
<?php
$numero = $_REQUEST['numero'];

if (is_numeric($numero)) {
    print "<p>Ha escrito un número: $numero.</p>\n";
} else {
    print "<p>NO ha escrito un número: $numero.</p>\n";
}
?>
```

Escriba algo:

Funciones ctype_

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

Estas funciones se pueden aplicar a los datos recibidos de un formulario ya que hacen comprobaciones de todos los caracteres de una cadena. Quizás la más útil es:

- la función `ctype_digit()`, que evalúa si todos los caracteres son dígitos, por lo que permite identificar los números enteros positivos (sin punto decimal ni signo negativo)

El ejemplo siguiente muestra el uso de la función `ctype_digit()`.

```
<?php
$numero = $_REQUEST['numero'];

if (ctype_digit($numero)) {
    print "<p>Ha escrito un entero positivo: $numero.</p>\n";
} else {
    print "<p>NO ha escrito un entero positivo: $numero.</p>\n";
}
?>
```

Escriba algo:

Funciones filter_

Las funciones filter se crearon como extensión PECL, pero se incluyeron en PHP 5.2 (2006).

La función filter más simple es la función [filter var\(\\$valor \[, \\$filtro \[, \\$opciones\]\]\)](#), que devuelve los datos filtrados o **false** si el filtro falla.

Los filtros predefinidos de validación son los siguientes:

Filtro	Tipo de datos
FILTER_VALIDATE_INT	entero
FILTER_VALIDATE_BOOLEAN	booleano
FILTER_VALIDATE_FLOAT	float
FILTER_VALIDATE_REGEXP	expresión regular
FILTER_VALIDATE_URL	URL
FILTER_VALIDATE_EMAIL	dirección de correo
FILTER_VALIDATE_IP	dirección IP
FILTER_VALIDATE_MAC	dirección MAC física



El problema de los filtros **FILTER_VALIDATE_INT** y **FILTER_VALIDATE_FLOAT** es que dan **false** si el argumento es 0, lo que complica su uso para detectar números porque el caso del 0 debe considerarse aparte al hacer la validación. Se supone que ese comportamiento se corrigió en PHP 5.4, pero no está claro que el problema esté resuelto en versiones posteriores.

El ejemplo siguiente muestra el uso de la función `filter_var()` con el argumento `FILTER_VALIDATE_INT`.

```
<?php
$numero = $_REQUEST['numero'];

if (filter_var($numero, FILTER_VALIDATE_INT)) {
    print "<p>Ha escrito un número entero: $numero.</p>\n";
} else {
    print "<p>NO ha escrito un número entero: $numero.</p>\n";
}
?>
```

Escriba algo:

Funciones `xxx_exists()`

- **Función `function_exists()`**

La función booleana `function_exists()` devuelve si la función existe o no.

Algunas de las funciones que se comentan en esta página no existen en versiones antiguas de PHP que todavía se utilizan, por lo que puede ser conveniente comprobar si una función existe antes de utilizarla.

El ejemplo siguiente muestra el uso de la función `function_exists()`.

Número:

```
<?php
print "<pre>"; print_r($_REQUEST); print "</pre>\n";

$numero = $_REQUEST['numero'];

if (function_exists('filter_var')) {
    if (filter_var($numero, FILTER_VALIDATE_INT)) {
        print "<p>Ha escrito el número entero: $numero.
</p>\n";
    } else {
        print "<p>NO ha escrito un número entero.
</p>\n";
    }
} else {
    print "<p>La función <strong>filter_var</strong> "
        . "no está disponible en este servidor.</p>\n";
}
?>
```

```
Array (
    [numero] => 2.5
)
La función filter_var no está disponible en este servidor.
```

- **Función `array_key_exists()`**

La función booleana `array_key_exists($índice, $matriz)` devuelve si un elemento determinado de una matriz existe o no.

Para comprobar si existe un elemento de una matriz también se puede utilizar la función `isset()`.

4. Glosario

ALGORITMO: Conjunto de sentencias / instrucciones en lenguaje nativo, los cuales expresan la lógica de un programa.

ALGORITMO CUALITATIVO: Son aquellos que resolver un problema no ejecuta operaciones matemática en el desarrollo de algoritmo.

ALGORITMO CUANTITATIVO: Son aquellos algoritmos que ejecutan operaciones numéricas durante su ejecución.

ARCHIVO: Son un conjunto de registros lógicos.

BASE DE DATOS: Es un almacenamiento colectivo de las bibliotecas de datos que son requeridas y organizaciones para cubrir sus requisitos de procesos y recuperación de información.

BASIC: (BEGINNERS ALL PURPOSE SYMBOLIC INSTRUCTION CODE), Lenguaje de instrucciones simbólicas de propósito general para principiantes, está disponible en modo compilador e interprete, siendo este último el más popular para el usuario circunstancial y para el programador principiante.

BIT:(dígito binario) un dígito simple de un numero binario (1 ó 0) en el ordenador.

BUFFERS: Memoria intermedia, una porción reservada de la memoria, que se utiliza para almacenar datos mientras son procesados.

BYTE: Grupo de bits adyacentes operados como una unidad. (grupos de 8 bits).

CAMPO: Es el espacio en la memoria que sirve para almacenar temporalmente un dato durante el proceso, Su contenido varia durante la ejecución del programa.

CAMPO NUMÉRICO: el que solo puede almacenar valores (dígitos).

CAMPO ALFANUMERICO: el que puede almacenar cualquier carácter (dígito, letra, símbolo especial).

CODIFICACIÓN: Es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por el ordenador, la serie de instrucciones detalladas se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

COMPILADOR: Programa de ordenador que produce un programa en lenguaje de máquina, de un programa fuente que generalmente está escrito por el programador en un lenguaje de alto nivel.

CONSTANTE: Valor o conjunto de caracteres que permanecen invariables durante la ejecución del programa.

DATO: El término que usamos para describir las señales con las cuales trabaja el ordenador es dato; Aunque las palabras dato e información muchas veces son usada indistintamente, si existe una diferencia importante entre ellas. En un sentido estricto, los datos son las señales individuales en bruto y sin ningún significado que manipulan los ordenadores para producir información.

DIAGRAMA DE FLUJO: Es la representación gráfica de una secuencia de instrucciones de un programa que ejecuta un ordenador para obtener un resultado determinado.

DOCUMENTACIÓN: Es la guía o comunicación escrita es sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas. A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a

comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento).

La documentación de un programa se divide en interna, externa y de usuario final.

EDITOR: Es un software empleado para crear y manipular archivos de texto, tales como programas en lenguaje fuente, lista de nombres y direcciones.

FREEWARE: (Software gratis del inglés free software, aunque esta denominación también se confunde a veces con "libre" por la ambigüedad del término en el idioma inglés) define un tipo de software que se distribuye sin costo, disponible para su uso y por tiempo ilimitado, siendo una variante gratuita del shareware, en el que la meta es lograr que un usuario pruebe el producto durante un tiempo ("trial") limitado, y si le satisface, pague por él, habilitando toda su funcionalidad.

LENGUAJE DE ALTO NIVEL: Los lenguajes de programación de alto nivel (BASIC, pascal, cobol, fortran, etc.) son aquellos en los que las instrucciones o sentencias al ordenador son escritas con palabras similares a los lenguajes humanos (en general en inglés), lo que facilita la escritura y comprensión del programa.

LENGUAJE DE BAJO NIVEL (ENSAMBLADOR): En este lenguaje las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos para las operaciones y direcciones simbólicas.

LENGUAJE MÁQUINA: Son aquellos cuyas instrucciones son directamente entendibles por el ordenador y no necesitan traducción posterior para que la CPU pueda comprender y ejecutar el programa. Las instrucciones en lenguaje maquina se expresan en términos de la unidad de memoria más pequeña el bit (dígito binario 0 ó 1).

LENGUAJE DE PROGRAMACIÓN: Sistema de símbolos y reglas que permite la construcción de programas con los que el ordenador puede operar así como resolver problemas de manera eficaz. Estos contienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada / salida, calculo, manipulación de textos, lógica / comparación y almacenamiento / recuperación.

MySQL: Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones.

OPERADORES: Un operador es un símbolo que indica al compilador que realice manipulaciones lógicas o matemáticas específicas. Los operadores del mismo nivel de precedencia son evaluados por el compilador de izquierda a derecha. Por supuesto, se puede utilizar paréntesis para ordenar la evaluación. También, conviene utilizar paréntesis para hacer más claro el orden en que se producen las evaluaciones, tanto para la persona que lo elabora o para los que después tengan que seguir el programa.

PROGRAMA: Es una colección de instrucciones que indican al ordenador que debe hacer. Un programa se denomina software, por lo tanto, programa, software e instrucción son sinónimos.

PROGRAMA FUENTE: Instrucción escrita por el programador en un lenguaje de programación para plantear al ordenador el proceso que debe ejecutar.

PROGRAMACIÓN ESTRUCTURADA: Método disciplinado de escribir programas que sean claros, que se demuestren que son correctos y fáciles de modificar

PROGRAMADOR: Un individuo que diseña la lógica y escribe las líneas de código de un programa de ordenador.

PRUEBA Y DEPURACIÓN: Los errores humanos dentro de la programación de ordenadores son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama depuración. La prueba consiste en la captura de datos hasta que el programa no presente errores (los más comunes son los sintácticos y lógicos).

PSEUDOCÓDIGO: Mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencial, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.

El pseudocódigo se concibió para superar las dos principales desventajas del Diagrama de Flujo: el diagrama de flujo es lento de crear y difícil de modificar sin un nuevo redibujo. Por otra parte el pseudocódigo es más fácil de utilizar ya que es similar al lenguaje natural.

VARIABLE: En programación es una estructura que contiene datos y recibe un nombre único dado por el programador, mantiene los datos asignados a ella hasta que un nuevo valor se le asigne o hasta que el programa termine.

5. Bibliografía

<http://tutorialphp.net/iniciacion-a-php-7/introduccion-a-php-7/>

<http://php.net/manual/es/index.php>

<http://www.ww.mclibre.org/consultar/php/>

http://administraciondesistemas.pbworks.com/f/Manual_PHP5_Basico.pdf

<https://www.mundomanuales.com/manuales/3144.pdf>

<https://www.taringa.net/posts/info/16954106/Glosario-de-Lenguajes-de-Programacion-y-Base-de-datos.html>

<http://php.net/manual/es/history.php.php>

<https://desarrolloweb.com/articulos/436.php>

6. Anexos

Manual de Programación en PHP.pdf

Manual imprescindible PHP5.pdf

Manual_PHP5_Basico.pdf

PHP Manual Completo Español.pdf

7. Cuestionario de autoevaluación.

1. ¿Cuál de los siguientes requisitos para que un control envíe información es incorrecto?

- a) Que el control esté incluido en un formulario.
- b) Que el formulario tenga establecido el atributo “method”.
- c) que el formulario contenga un botón de tipo submit

2. El programa que recibe los datos, los guarda automáticamente en:

- a) La Base de Datos ubicada en la nube.
- b) En la matriz \$_HPBOOK.
- c) En la matriz \$_REQUEST

3. Cuando, en un formulario con “get”, existen varios controles que envíen información, los valores aparecen separados por:

- a) Comillas (“”).
- b) El carácter ampersand (&).
- c) El carácter dólar (\$).

4. El control “input submit, button”, se puede crear con las siguientes etiquetas:

- a) <input> .
- b) <button>.
- c) Ambas respuestas son correctas.

5. El control “input checkbox”...

- a) Se envía siempre.
- b) Se envía solamente si se marca la casilla.
- c) Sólo se envía si la casilla no tiene definido el atributo “value”.

6. Si el control “menú” (o “select”) admite selección múltiple, entonces, el nombre de dicho menú debe acabar con:

- a) Paréntesis ().
- b) Comillas ””.
- c) Corchetes []

7. Si atributo “name” de la matriz \$_REQUEST contiene espacios, estos se sustituyen por:

- a) Guiones bajos (_).
- b) Guiones medios (-).
- c) Ninguna de las anteriores respuestas es correcta.

8. Si se hace referencia a los elementos de la matriz \$_REQUEST...

- a) Si la referencia se hace dentro de una cadena, el índice se debe escribir con comillas.
- b) Si la referencia se hace fuera de una cadena, el índice se debe escribir sin comillas
- c) Si la referencia se hace dentro de una cadena, el índice se debe escribir sin comillas.

9. ¿Qué función es la adecuada para eliminar los espacios en blanco que, por error, se hayan podido introducir en un formulario?

- a) La función trim(\$cadena).
- b) La función isset().
- c) La función strip_tags()

10. ¿Qué función trabaja como una solución de modo general cuando aparecen problemas con caracteres especiales como "", (), &, <>?

- a) La función strip_tags.
- b) La función htmlspecialchars
- c) La función (").