

Bloque B

Páginas Web Dinámicas

Unidad 6. Obteniendo datos desde el navegador

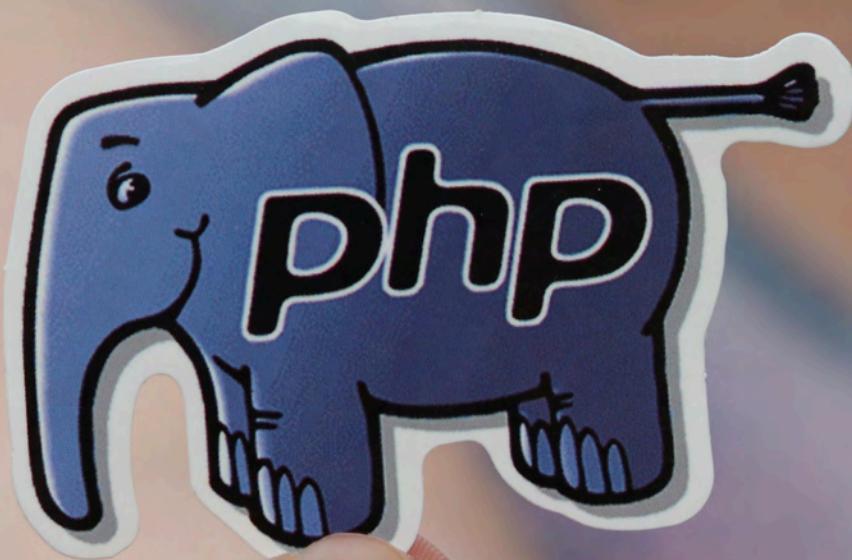


Contenidos

1. Introducción
2. Obtención de Datos desde el Navegador a través de Métodos HTTP
3. Validación y Gestión de Errores en Datos
4. Saneamiento y Escape de Datos
5. Envío de Datos desde Formularios
6. Comprobación y Validación de Formularios

Contenidos

7. Validación Específica de Datos: Números y Texto
8. Validación de Selecciones y Opciones en Formularios
9. Uso de Filtros para Validación y Saneamiento de Datos
10. Filtros sobre variables: Flags y Opciones Avanzadas
11. Filtros de Saneamiento de Datos
12. Actividad final
13. Resumen



1. Introducción

Introducción

En esta unidad, aprenderás cómo acceder a los datos que los navegadores envían al intérprete PHP, asegurarte de que están listos para ser utilizados y de que son seguros para ser mostrados en páginas web dinámicas.

Introducción

En la introducción a este bloque, vimos que las páginas HTML tienen dos mecanismos para enviar datos al servidor: añadir información a los enlaces o proporcionar formularios para llenar.

También vimos cómo estos datos se envían a través de HTTP GET (en una cadena de consulta) o HTTP POST (en las cabeceras HTTP que se envían con cada petición de página).

Introducción

En esta unidad, aprenderás cómo acceder a esos datos para poder utilizarlos en la página.

Esto implica cuatro pasos clave:

- **Recoger** cada dato de la cadena de consulta o de las cabeceras HTTP.
- **Validar** cada dato para comprobar que se ha proporcionado un valor y que está en el formato correcto (por ejemplo, si una página necesita un número, se comprueba que se ha proporcionado un número, no texto).
- **Decidir** si la página puede o no procesar los datos facilitados por el visitante. Si no es así, puede ser necesario mostrar mensajes de error al visitante.
- Escapar o **sanear** los datos para asegurarnos de que se pueden utilizar con seguridad en la página, ya que ciertos caracteres pueden impedir que una página se muestre correctamente o incluso causar daños al sitio.

Introducción

No existe una forma estándar de realizar cada uno de estos cuatro pasos; diferentes desarrolladores utilizan diferentes enfoques.

Esta unidad presenta una selección de las diferentes formas de recopilar datos y asegurarse de que su utilización es segura.

4 pasos para recolectar y usar datos

Hay cuatro pasos para recopilar datos de los visitantes y asegurarnos de que se pueden utilizar con seguridad. Hay diferentes formas de lograr cada uno de estos pasos y conoceremos varias de ellas en esta unidad.

1. Recopilar datos

En primer lugar, se recogen los datos que los navegadores envían al servidor. Puedes hacer esto utilizando:

- Dos **arrays superglobales** que el intérprete PHP crea cada vez que se solicita un archivo PHP.
- Dos funciones incorporadas llamadas **funciones de filtro (filter functions)**.

4 pasos para recolectar y usar datos

1. Recopilar datos

Como se verá, una página no siempre recibirá los valores que necesita para realizar su tarea, y esto puede causar un error.

- Si un dato es opcional, se puede especificar un valor por defecto que la página debe utilizar si no se suministra.
- Si no es opcional, y faltan datos, puede que tengas que decirle al visitante que no ha proporcionado suficiente información.

4 pasos para recolectar y usar datos

2. Validar datos

Una vez que una página PHP ha recolectado datos de un navegador, a menudo **validará** cada pieza individual de datos que recibió para asegurar que no causará errores cuando la página se ejecute. Esto implica comprobar:

- Si la página tiene los datos que necesita para realizar su tarea. Esto se conoce como **datos requeridos**.
- Si los datos están en el **formato correcto**. Por ejemplo, si su página necesita un número para realizar un cálculo, puede comprobar que ha recibido un número. O, si espera recibir una dirección de correo electrónico, se puede verificar que el texto está en el formato correcto para ser una dirección de correo electrónico válida.

4 pasos para recolectar y usar datos

2. Validar datos

PHP proporciona dos maneras de validar datos; puedes:

- Escribir tus propias funciones definidas por el usuario.
- Utilizar un conjunto de filtros incorporados con las funciones de filtro. Cada filtro valida diferentes tipos de datos.

4 pasos para recolectar y usar datos

3. Decidir la acción

Una vez que una página ha recopilado y validado todos los valores individuales que necesita, puede determinar si tiene o no todos los datos que necesita para ejecutarse:

- Si todos los datos son válidos, se pueden procesar.
- Si alguno de los datos no es válido o falta, no debe utilizarse. En su lugar, la página puede mostrar al usuario un mensaje de error.

4 pasos para recolectar y usar datos

3. Decidir la acción

El proceso de mostrar errores cuando los datos no son válidos es ligeramente diferente para los formularios que para las cadenas de consulta.

- Si los datos del formulario no son válidos, puedes volver a mostrar el formulario con mensajes junto a cualquier control de formulario que haya proporcionado datos no válidos. Los mensajes deben indicar al usuario cómo proporcionar los datos en el formato correcto.
- Si una cadena de consulta tiene datos incorrectos, no se debe esperar que los visitantes editen la cadena de consulta. En su lugar, se debe proporcionar un mensaje que explique cómo el usuario puede solicitar los datos que quería.

4 pasos para recolectar y usar datos

4. Escapar o sanear los datos

Cada vez que se muestran datos que un visitante ha proporcionado en una página, es necesario **escaparlos** para asegurarse de que su visualización es segura. Esto implica sustituir un conjunto de caracteres que los navegadores tratan como código (como los símbolos < y >) por unas cosas llamadas **entidades**. Las entidades le dicen al navegador que muestre esos caracteres (en lugar de ejecutarlos como código HTML).

Si no se realiza este paso antes de mostrar los datos en una página, un pirata informático podría intentar que la página ejecutara un archivo JavaScript malicioso.

4 pasos para recolectar y usar datos

4. Escapar o sanear los datos

Si el usuario proporciona datos que luego se utilizan en una URL, también hay que escapar cualquier carácter que tenga un significado especial (como barras inclinadas y signos de interrogación). Si no se escapan estos caracteres, es posible que el servidor web no pueda procesar la URL.



2. Obtención de Datos desde el Navegador a través de Métodos HTTP

Obtención de datos enviados a través de HTTP GET

Cuando se añaden datos a una cadena de consulta al final de una URL, el intérprete de PHP añade esos datos a un array superglobal llamado `$_GET` para que el código PHP de la página pueda acceder a ellos.

Obtención de datos enviados a través de HTTP GET

A continuación se muestra un enlace HTML. En su atributo href, puede ver la URL de la página a la que enlaza.

Al final de la URL, hay una cadena de consulta que contiene dos pares nombre/valor que se envían al servidor cuando el visitante hace clic en el enlace.

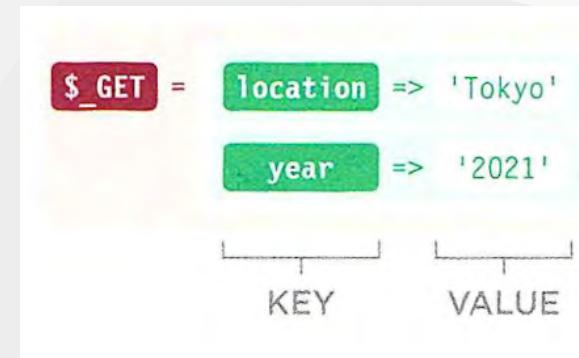


Obtención de datos enviados a través de HTTP GET

Cuando el intérprete de PHP recibe esta petición, añade los datos de la cadena de consulta a un array superglobal llamado `$_GET`.

Como todos los arrays superglobales que genera el intérprete de PHP, `$_GET` es un array asociativo. Se le da un elemento por cada par nombre/valor que está en la cadena de consulta, donde en cada elemento:

- La **clave** es el nombre que se envía.
- El **valor** es el valor enviado con el nombre.



Obtención de datos enviados a través de HTTP GET

El código en el archivo PHP puede acceder a los valores en el array superglobal `$_GET` de la misma manera que accedería a los valores de cualquier array asociativo:

```
$location = $_GET['location'];
```

VARIABLE

KEY

Obtención de datos enviados a través de HTTP GET

A menudo, un único archivo PHP se utiliza para mostrar varias páginas de un sitio web, y los datos de la cadena de consulta se utilizan para determinar qué datos se muestran en la página.

En el siguiente ejemplo, un array tiene tres elementos. Cada elemento contiene la ciudad y la dirección de una tienda. Un valor en la cadena de consulta selecciona qué datos de la tienda deben mostrarse, por lo que este archivo PHP crea tres páginas del sitio; cada una es para una tienda diferente. Los datos del array también se utilizan para crear los enlaces que solicitan estas tres páginas.

Ejemplo: Usando una cadena de consulta para seleccionar contenido

Este ejemplo recoge el nombre de una ciudad de la cadena de consulta y muestra la dirección de una tienda en esa ciudad.

1. La variable `$cities` contiene un array asociativo. Cada clave es el nombre de una ciudad diferente; cada valor es la dirección de una sucursal de la tienda en esa ciudad.
2. El nombre de la ciudad se recoge del array superglobal `$_GET` y se almacena en una variable llamada `$city` (Ten en cuenta que distingue entre mayúsculas y minúsculas).
3. El nombre de la ciudad se utiliza para seleccionar la dirección de la sucursal en esa ciudad del array creado en el paso 1, y se almacena en una variable llamada `$address`.

Ejemplo: Usando una cadena de consulta para seleccionar contenido

4. Un bucle `foreach` recorre cada elemento del array `$cities`.
5. Dentro del bucle, se crea un enlace para cada ciudad. El nombre de la ciudad se escribe en la cadena de consulta, y de nuevo como texto del enlace. Esto muestra como PHP puede crear enlaces, y como esos enlaces pueden apuntar a un solo archivo que puede mostrar diferentes datos.
6. Los valores que fueron almacenados en las variables `$city` y `$address` en los Pasos 2 y 3 son mostrados en la página.

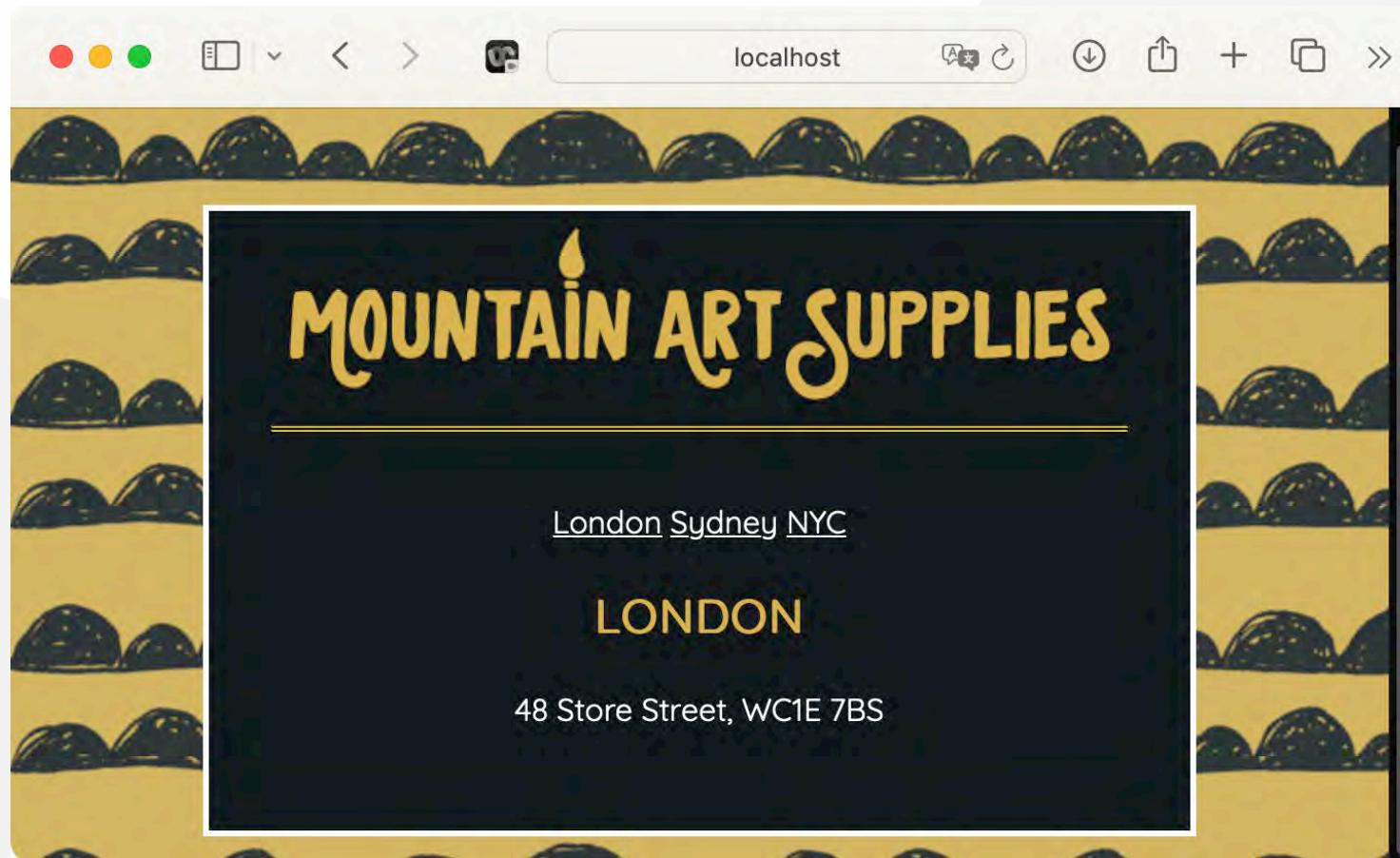
Ejemplo: Usando una cadena de consulta para seleccionar contenido

```
<?php
    $cities = [
        'London' => '48 Store Street, WC1E 7BS',
        'Sydney' => '151 Oxford Street, 2021',
        'NYC'     => '1242 7th Street, 10492',
    ];
    $city    = $_GET['city'];
    $address = $cities[$city];
?>

...
<?php foreach ($cities as $key => $value) { ?>
    <a href="get-1.php?city=<?= $key ?><?= $key ?></a>
<?php } ?>

<?php
    <h1><?= $city ?></h1>
    <p><?= $address ?></p>
```

Ejemplo: Usando una cadena de consulta para seleccionar contenido



Ejemplo: Usando una cadena de consulta para seleccionar contenido

Practica cómo obtener datos desde una página PHP utilizando la cadena de consulta y el array superglobal `$_GET`. Puedes tomar como base el ejemplo anterior.

Instrucciones:

- 1. Crear el archivo principal `index.php`**: Este archivo será el punto de entrada de la aplicación y mostrará una lista de productos. Cada producto será un enlace que llevará a una página de detalles del producto.
- 2. Crear el archivo de detalles del producto `product.php`**: Este archivo recibirá el nombre del producto desde la cadena de consulta y mostrará su descripción.

Ejemplo: Usando una cadena de consulta para seleccionar contenido

3. Realiza las siguientes tareas

- Ejecuta `index.php` en tu servidor web local.
- Haz clic en los enlaces de productos para ver los detalles de cada producto.
- Modifica los arrays `$products` en ambos archivos para agregar más productos y descripciones.
- Experimenta agregando más detalles al producto, como el precio o la disponibilidad, y muestralos en `product.php`.

Gestión de datos ausentes en los arrays superglobales

Si se intenta acceder a una clave que no ha sido añadida a un array superglobal, el intérprete de PHP emite un error.

Para evitar estos errores, se puede comprobar si la clave está en el array superglobal antes de acceder a ella.

Cuando alguien comparte un enlace a una página, puede omitir accidentalmente parte o toda su cadena de consulta.

Gestión de datos ausentes en los arrays superglobales

En el ejemplo anterior, se puede comprobar que si a la cadena de consulta le faltan datos, no puede ser añadida al array superglobal `$_GET`. Si un archivo PHP intenta acceder a esos datos, el intérprete PHP genera un error que dice *Undefined array key* o *Undefined index* porque está intentando acceder a una clave (o índice) que no ha sido añadida al array superglobal `$_GET`.

Para evitar este error, las páginas deberían comprobar si se ha añadido un valor al array superglobal `$_GET` antes de intentar acceder a él.

Gestión de datos ausentes en los arrays superglobales

PHP tiene una función incorporada llamada `isset()` que acepta un nombre de variable, una clave de un array, o una propiedad de un objeto como argumento. Devuelve verdadero si esa variable, clave o propiedad existe, y su valor no es nulo. En caso contrario, devuelve false. Es importante destacar que no provocará un error si la variable, clave o parámetro especificado no existe.

A continuación, se declara una variable llamada `$city`. Un operador ternario comprueba si la superglobal `$_GET` tiene una clave llamada ciudad y su valor no es nulo. Si lo tiene, se almacenará en la variable `$city`. En caso contrario, `$city` contendrá una cadena en blanco.

Gestión de datos ausentes en los arrays superglobales

```
$city = isset($_GET['city']) ? $_GET['city'] : '';
```

VARIABLE DOES KEY EXIST? YES: STORE ITS VALUE NO: STORE BLANK STRING

Gestión de datos ausentes en los arrays superglobales

PHP 7 introdujo el **operador de coalescencia nula** `??` que actúa como una forma abreviada de utilizar `isset()` en la condición de un operador ternario.

Si no existe un valor a la izquierda del operador de coalescencia nula, se suministra un valor alternativo que debe utilizarse a la derecha del mismo.

```
$city = $_GET['city'] ?? '';
```

VARIABLE TRY TO STORE THIS VALUE IF IT DOES NOT EXIST: STORE BLANK STRING

Ejemplo: Usando una cadena de consulta para seleccionar contenido (II)

Este ejemplo se basa en el anterior (ver [Ejemplo: Usando una cadena de consulta para seleccionar contenido](#)). Se destacan las diferencias.

1. El valor almacenado en la variable `$city` se asigna utilizando el operador de coalescencia nula. Si el array superglobal `$_GET` :
 - Tiene una clave llamada ciudad y su valor no es nulo, su valor se almacenará en la variable `$city` .
 - No tiene una clave llamada ciudad, o su valor es nulo, se almacenará una cadena vacía en la variable `$ciudad` .

Ejemplo: Usando una cadena de consulta para seleccionar contenido (II)

2. La variable `$city` se utiliza en la condición de una sentencia `if`. Si el valor es una cadena que no está vacía, el intérprete de PHP trata ese valor como verdadero y ejecuta el bloque de código subsiguiente.
3. La variable `$address` almacena la dirección de la sucursal en la ciudad que fue nombrada en la cadena de consulta.
4. En caso contrario, si el valor de la variable `$city` es una cadena vacía, se ejecuta el segundo bloque de código.
5. La variable `$address` almacena un mensaje indicando al visitante que seleccione una ciudad.

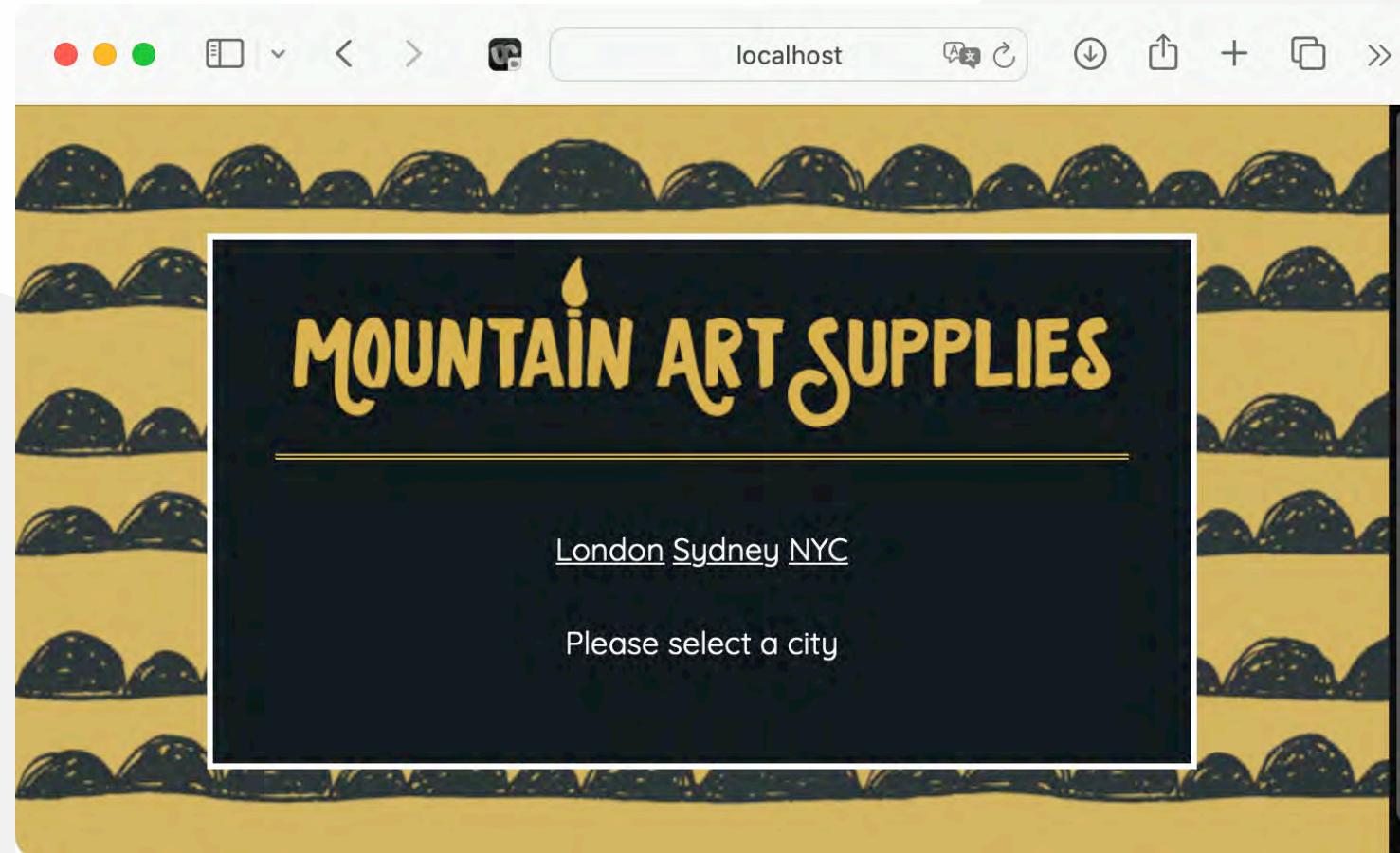
Ejemplo: Usando una cadena de consulta para seleccionar contenido (II)

```
<?php
$cities = [
    'London' => '48 Store Street, WC1E 7BS',
    'Sydney' => '151 Oxford Street, 2021',
    'NYC'     => '1242 7th Street, 10492',
];
① $city = $_GET['city'] ?? '';
② if ($city) {
③     $address = $cities[$city];
④ } else {
⑤     $address = 'Please select a city';
}
?>

***<?php foreach ($cities as $key => $value) { ?>
    <a href="get-2.php?city=<?= $key ?>"><?= $key ?></a>
<?php } ?>

<h1><?= $city ?></h1>
<p><?= $address ?></p>
```

Ejemplo: Usando una cadena de consulta para seleccionar contenido (II)



Ejemplo: Usando una cadena de consulta para seleccionar contenido (II)

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- En la cadena de consulta, utiliza Tokio como ciudad. La página mostrará un error porque no encuentra esa clave en el array `$cities`.
- Añade un nuevo elemento al array del paso 1 con la clave Tokio y añade una dirección para él, luego intenta utilizarlo de nuevo en la cadena de consulta.



3. Validación y Gestión de Errores en Datos

Validando datos

Antes de que una página PHP utilice los datos que ha recopilado, éstos deben validarse para garantizar que no causarán errores cuando se ejecute la página.

Validar los datos que recibe una página implica comprobar que el fichero PHP tiene:

- Los datos necesarios para realizar una tarea, lo que se conoce como **datos requeridos**.
- Los datos en el **formato correcto**. Por ejemplo, si una página necesita un número para realizar un cálculo, puede comprobar que ha recibido un número (en lugar de una cadena).

Validando datos

En el ejemplo anterior, la cadena de consulta necesitaba un:

- Nombre y valor para especificar la tienda a mostrar
- Valor que coincidiera con una clave del array de ciudades

Si el valor suministrado en la cadena de consulta no se encuentra en el array de ciudades, el intérprete de PHP emite un error. Por lo tanto, antes de intentar mostrar la ciudad en la página, el código puede comprobar si el valor en la cadena de consulta está presente en el array de ciudades.

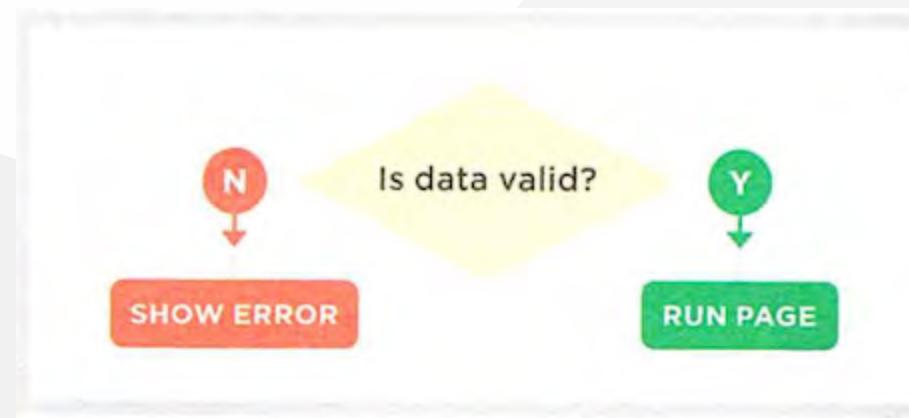
Validando datos

Aprenderemos varias formas de validar diferentes tipos de datos a lo largo del resto de esta unidad.

En el ejemplo siguiente se utiliza la función `array_key_exists()` de PHP (que ya conocimos en la unidad anterior) para comprobar si el valor de la cadena de consulta coincide con una clave del array de ciudades. La función devuelve *true* si se encuentra la clave, *false* si no; y el valor que devuelve la función se almacena en una variable llamada `$valid`.

Validando datos

Una vez validados los datos, la página debe determinar si debe ejecutar o no el resto del código.



Validando datos

El ejemplo utilizado hasta ahora en la unidad se amplía a continuación. La variable `$valid` se utiliza en la condición de una sentencia `if` para determinar si la página puede o no procesar los datos:

- Si los datos son válidos, la página puede obtener la ubicación del almacén del array y mantenerla en una variable llamada `$address` lista para mostrarla más tarde en la página.
- Si los datos no son válidos, la variable `$address` almacena un mensaje indicando al visitante que seleccione una ciudad. Esto proporciona información útil al visitante indicándole cómo utilizar la página y obtener la información que está buscando.

Más adelante en esta unidad, aprenderás cómo tratar con páginas que necesitan recoger múltiples valores del navegador, y cómo comprobar si todos los valores son válidos o no.

Ejemplo: validación de datos en la cadena de consulta

Este ejemplo se basa en los anteriores y utiliza la validación para comprobar si la cadena de consulta contiene una ubicación válida.

1. Si la cadena de consulta contiene una ciudad, se almacenará en una variable llamada `$city`. Si no, `$city` contendrá una cadena en blanco.
2. La función `array_key_exists()` comprueba si el valor de `$city` es una clave del array `$cities`. Si lo es, la variable `$valid` tendrá el valor `true`. En caso contrario, `$valid` tendrá el valor `false`.

Ejemplo: validación de datos en la cadena de consulta

3. La variable `$valid` se utiliza en la condición de una sentencia `if`. Si el valor que almacena es verdadero, se ejecutará el primer bloque de código.
4. La dirección de esa ciudad se recoge del array `$cities` y se almacena en la variable `$address`.
5. Si el valor de `$valid` es falso, se ejecuta el segundo bloque de código.
6. La variable `$address` contiene un mensaje que indica al visitante que seleccione una ciudad.

Ejemplo: validación de datos en la cadena de consulta

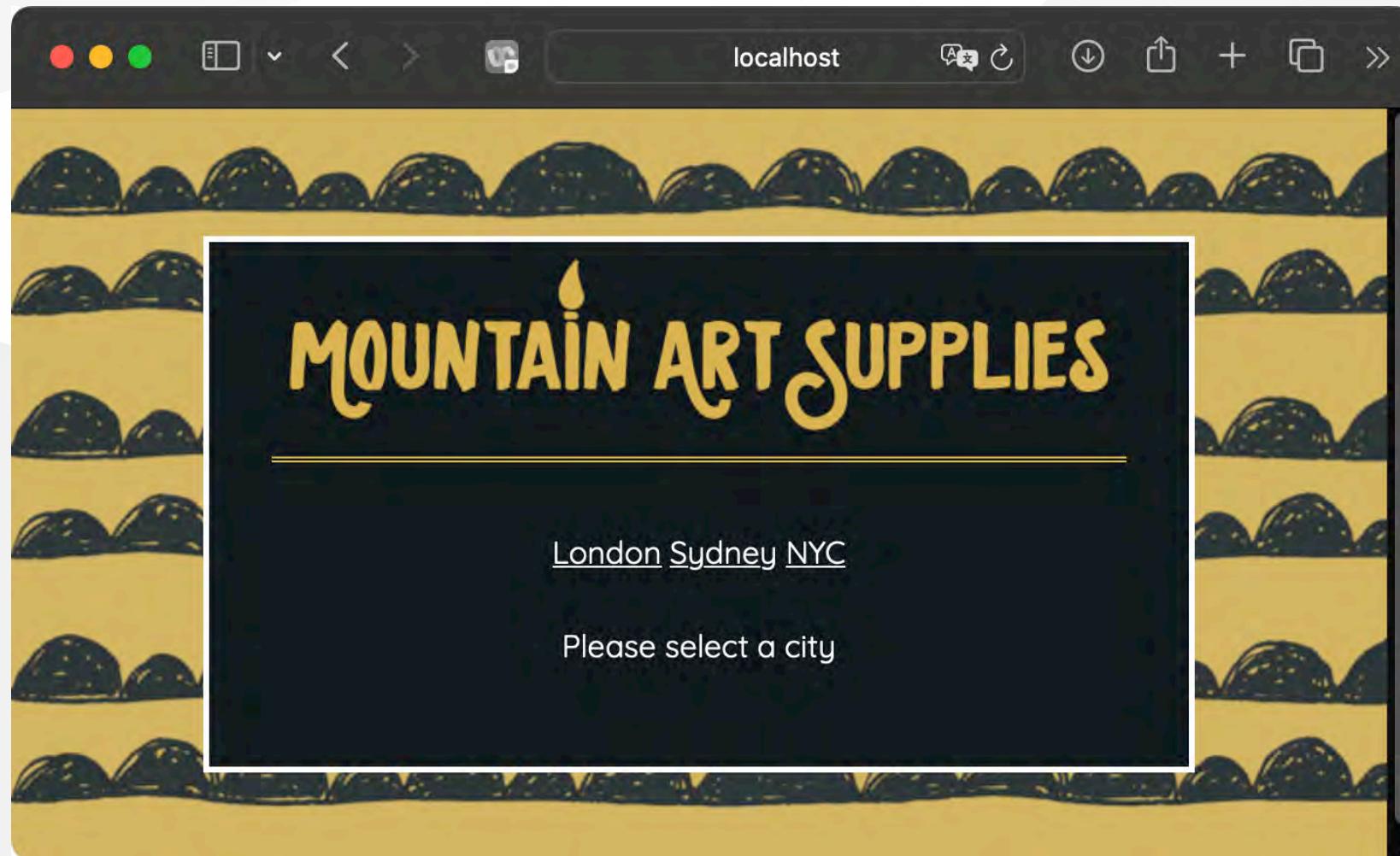
```
<?php
$cities = [
    'London' => '48 Store Street, WC1E 7BS',
    'Sydney' => '151 Oxford Street, 2021',
    'NYC'     => '1242 7th Street, 10492',
];
① $city = $_GET['city'] ?? '';
② $valid = array_key_exists($city, $cities);

③ if ($valid) {
④     $address = $cities[$city];
⑤ } else {
⑥     $address = 'Please select a city';
}
?>

...
<?php foreach ($cities as $key => $value) { ?>
    <a href="get-3.php?city=<?= $key ?>"><?= $key ?></a>
<?php } ?>

<h1><?= $city ?></h1>
<p><?= $address ?></p>
```

Ejemplo: validación de datos en la cadena de consulta



Ejemplo: validación de datos en la cadena de consulta

Tomando como ejemplo la actividad propuesta anterior (ver [Ejemplo: Usando una cadena de consulta para seleccionar contenido \(II\)](#)), realiza la siguiente tarea:

- Agrega validación adicional en `product.php` para manejar casos donde el parámetro `product` no está presente o no es válido.

Mostrar una página de error si faltan datos

Si una página necesita obtener datos de la cadena de consulta, pero esos datos faltan o no son válidos, el intérprete PHP puede decirle al navegador que solicite un archivo diferente que contenga un mensaje de error.

Mostrar una página de error si faltan datos

Validar los datos en la cadena de consulta es importante porque, cuando la gente se enlaza a tu sitio, es fácil que accidentalmente omitan datos de la cadena de consulta.

No se debe esperar que los visitantes sean capaces de editar los datos de la cadena de consulta, por lo que, si los datos no son válidos, se les puede ayudar:

- Mostrándoles un mensaje en la página. Esto podría decirles que la página que solicitaron no se pudo encontrar, o podría decirles que seleccionen de una lista de opciones (como el ejemplo de la página anterior).
- Enviarlos a una página diferente que contenga un mensaje de error.

Mostrar una página de error si faltan datos

Nótese cómo la condición en el código siguiente comprueba si los datos no son válidos verificando si el valor almacenado en `$valid` no es verdadero.

```
IF NOT VALID → if (!$valid) {  
    SET RESPONSE CODE → http_response_code(404);  
    REDIRECT TO ERROR PAGE → header('Location: page-not-found.php');  
    STOP CODE RUNNING → exit;  
}
```

Mostrar una página de error si faltan datos

En la unidad 5, viste que la función incorporada `header()` de PHP puede ser utilizada para establecer la cabecera *Location* que el intérprete PHP envía al navegador. Esto le dice al navegador que solicite una página diferente.

Mostrar una página de error si faltan datos

Cuando una página no puede ser mostrada porque los datos no son válidos, es una buena práctica actualizar el código de respuesta que el intérprete PHP envía de vuelta al navegador.

Esto ayuda a evitar que los motores de búsqueda añadan URLs incorrectas a sus resultados de búsqueda.

Mostrar una página de error si faltan datos

La función incorporada de PHP `http_response_code()` se utiliza para establecer el código de respuesta HTTP. Su único argumento es el código de respuesta que debe ser utilizado. El envío de un código de respuesta *404* indica que la página solicitada no ha podido ser encontrada.

Una vez establecidos el código de respuesta y la cabecera, el comando `exit` detiene la ejecución de cualquier otro código de la página (ya que podría provocar un error).

Ejemplo: Redirigir a los visitantes a una página de error

Este ejemplo envía a los visitantes a una página de error si los datos de la cadena de consulta no son una ciudad válida.

1. La función `array_key_exists()` de PHP comprueba si el nombre de la ciudad recogido de la cadena de consulta es una de las claves del array de ciudades. La función devuelve verdadero si existe, y falso en caso contrario. Este valor se almacena en una variable llamada `$valid`.
2. La condición de una sentencia `if` comprueba si el valor almacenado en `$valid` no es verdadero. (El operador `!` indica que no debe ser verdadero.) Si es falso, se ejecuta el siguiente bloque de código.

Ejemplo: Redirigir a los visitantes a una página de error

3. La función `http_response_code()` de PHP indica al intérprete de PHP que envíe el código de respuesta 404 de vuelta al navegador, indicando que la página no ha podido ser encontrada.
4. La función `header()` de PHP indica al intérprete de PHP que añada una cabecera *Location* para indicar al navegador que solicite en su lugar un archivo llamado `page-not-found.php`.
5. El comando `exit` de PHP le dice al intérprete PHP que no ejecute más código en el archivo.

Cuando el valor en `$valid` es verdadero, los Pasos 3-5 son ignorados y la página es mostrada.

Ejemplo: Redirigir a los visitantes a una página de error

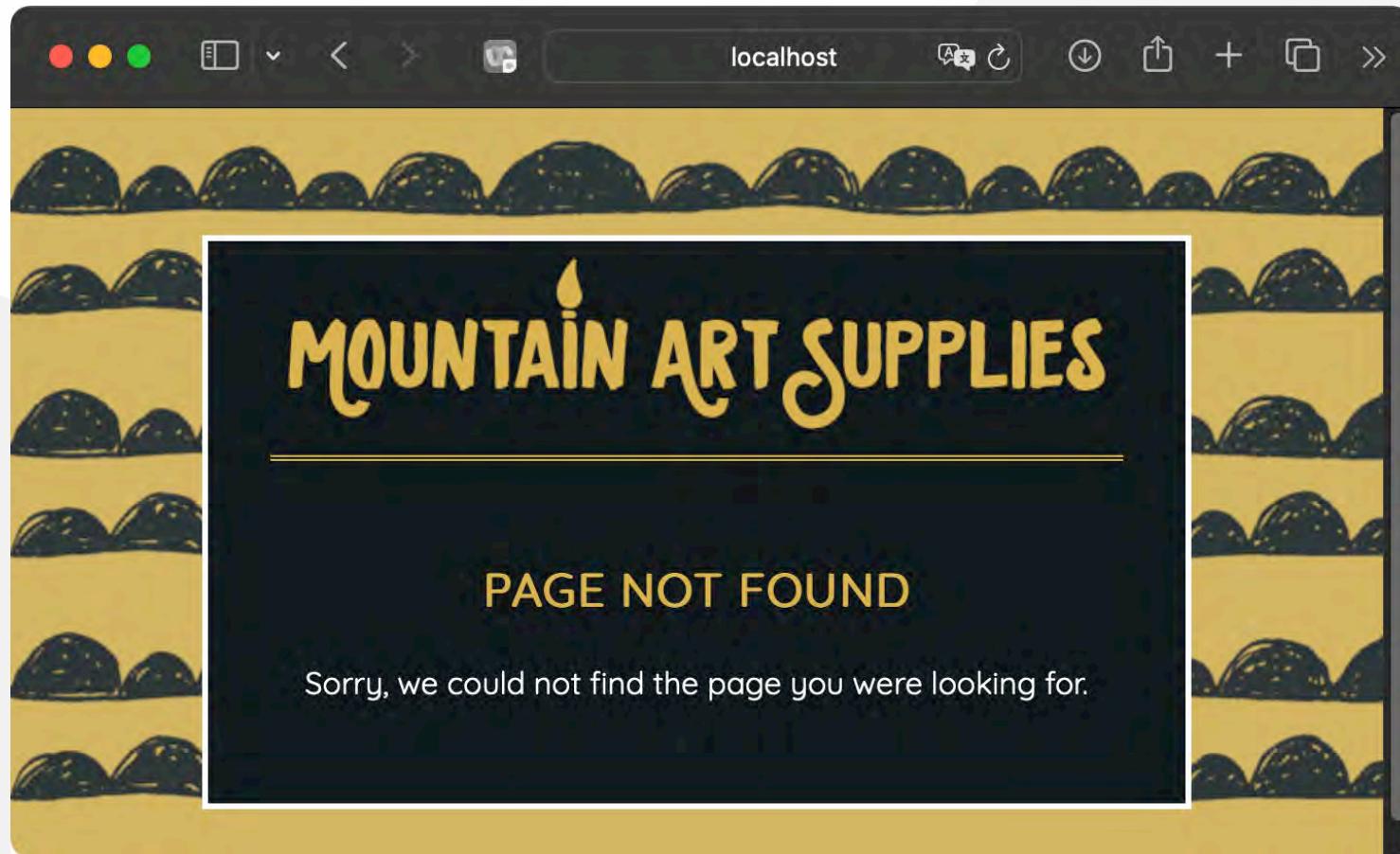
```
<?php
$cities = [
    'London' => '48 Store Street, WC1E 7BS',
    'Sydney' => '151 Oxford Street, 2021',
    'NYC'     => '1242 7th Street, 10492',
];
$city  = $_GET['city'] ?? '';
① $valid = array_key_exists($city, $cities);

② if (!$valid) {
③     http_response_code(404);
④     header('Location: page-not-found.php');
⑤     exit;
}
$address = $cities[$city];
?>

...
<?php foreach ($cities as $key => $value) { ?>
    <a href="get-4.php?city=<?= $key ?>"><?= $key ?></a>
<?php } ?>

<h1><?= $city ?></h1>
<p><?= $address ?></p>
```

Ejemplo: Redirigir a los visitantes a una página de error



Ejemplo: Redirigir a los visitantes a una página de error

Sobre la actividad propuesta anterior, implementa una nueva página php (p.e: `error-page.php`) que muestre un mensaje de error y redirige al usuario a dicha página cuando los datos proporcionados en la cadena de consulta no sean válidos.



4. Saneamiento y Escape de Datos

Escapando la salida

Cuando los valores que se han enviado al servidor se muestran en una página, deben **escaparse** para garantizar que los piratas informáticos no puedan utilizarlos para ejecutar secuencias de comandos maliciosas.

Escapando la salida

Escapar datos implica eliminar (y opcionalmente sustituir) cualquier carácter que no deba aparecer en un valor. Por ejemplo, HTML tiene cinco caracteres reservados que los navegadores tratan como código:

- < y > se utilizan en las etiquetas
- " y ' contienen valores de atributos
- & se utiliza para crear entidades

Escapando la salida

Para mostrar estos cinco caracteres en una página, deben sustituirse por un **nombre de entidad** o un **número de entidad** que los represente. Los navegadores muestran entonces los caracteres correspondientes en lugar de tratarlos como código.



Escapando la salida

Cuando una página recibe valores de un visitante y luego necesita mostrar esos valores en una página, debe comprobar si existen estos cinco caracteres reservados y sustituirlos por sus entidades. Esto puede hacerse utilizando la función integrada `htmlspecialchars()` de PHP (ver [Escape de caracteres HTML reservados](#)).

Si no se sustituyen los caracteres reservados de HTML por entidades de forma apropiada, los hackers podrían enviar valores que cargaran un archivo JavaScript con código malicioso. Es lo que se denomina un ataque **cross-site scripting (XSS)**.

Escapando la salida

Por ejemplo, si un visitante proporcionara el siguiente nombre de usuario y la página intentara mostrarlo, podría provocar la ejecución del script.

```
Luke<script src="http://eg.link/bad.js">
</script>
```

Cuando los caracteres reservados se sustituyen por entidades, los visitantes verían el texto anterior (y el script no se ejecutaría). En el código fuente HTML de la página, el nombre de usuario tendría el siguiente aspecto:

```
Luke&lt;script src="http://eg.link/bad.
js"&gt;&lt;/script&gt;
```

Escapando la salida

Los datos suministrados por los usuarios sólo deben aparecer en el marcado HTML visible en la página web (o en los elementos `<title>` o `<meta>`).

No se deben mostrar los datos suministrados por un usuario en:

- Comentarios en el código
- Reglas CSS (ya que pueden incluir un script en una página)
- Elementos `<script>`
- Nombres de etiquetas
- Nombres de atributos
- Como valor de atributos de eventos HTML como `onclick` y `onload`
- Como valor de un atributo HTML que carga archivos (como el atributo `src`)

Ejemplo: riesgo de no escapar la salida

Este ejemplo muestra lo que ocurre si no se escapan los datos.

1. Para los propósitos de este ejemplo, se muestra un enlace a esta misma página. El enlace tiene una cadena de consulta que contiene etiquetas <script> (En un ataque XSS real, el enlace a esta página podría aparecer en otro sitio web, en un correo electrónico o en otro tipo de mensaje).

Ejemplo: riesgo de no escapar la salida

2. La página PHP comprueba el array superglobal `$_GET` para ver si la cadena de consulta contiene un nombre llamado *msg*.
 - Si lo contiene, el valor correspondiente se almacena en una variable llamada `$message`.
 - Si no es así, `$message` almacena una instrucción indicando a los usuarios que hagan clic en el enlace.

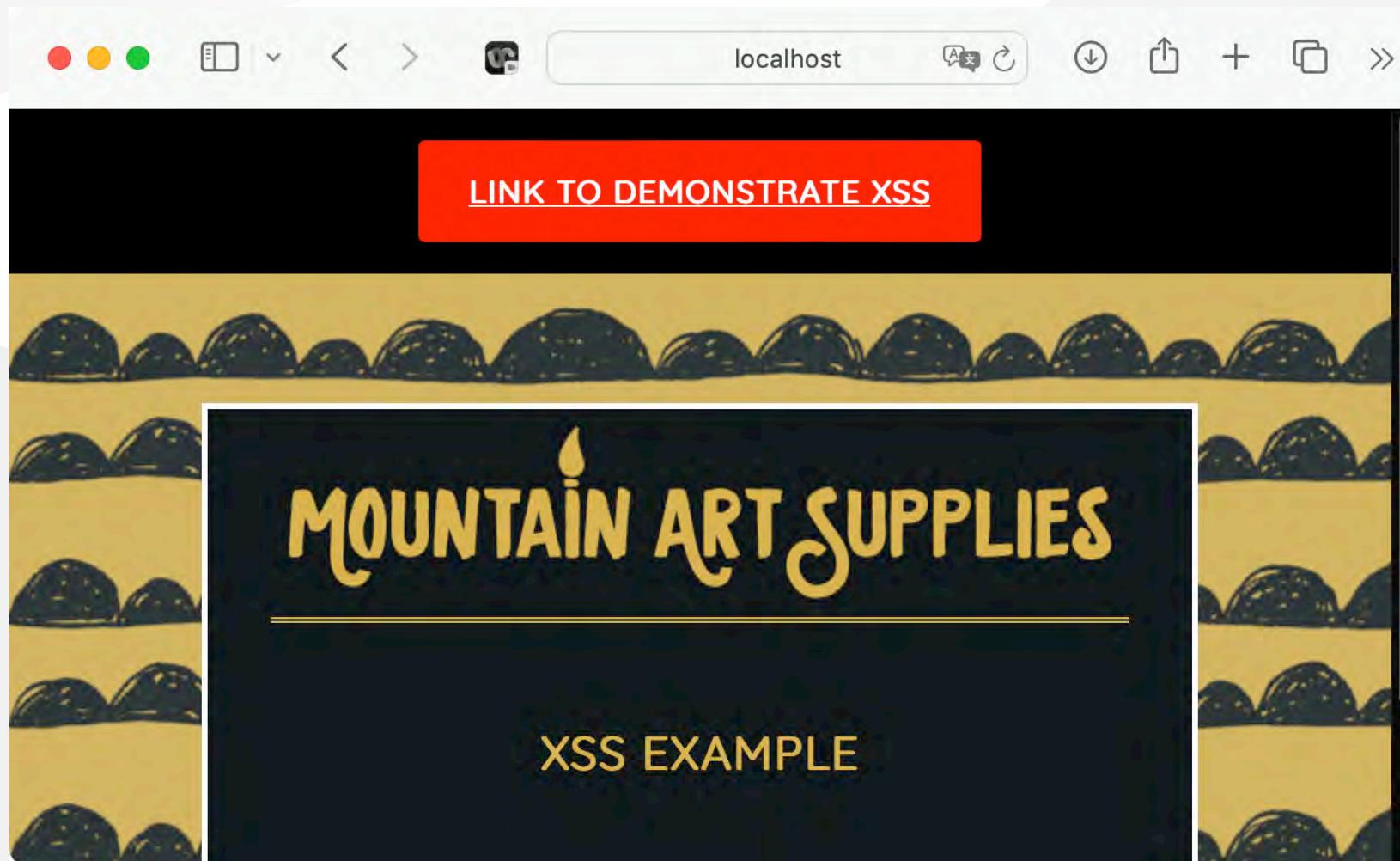
3. El valor en `$message` se muestra en la página.

Al hacer clic en el enlace de la parte superior de la página, se ejecutará el script porque no se ha escapado el valor de la cadena de consulta.

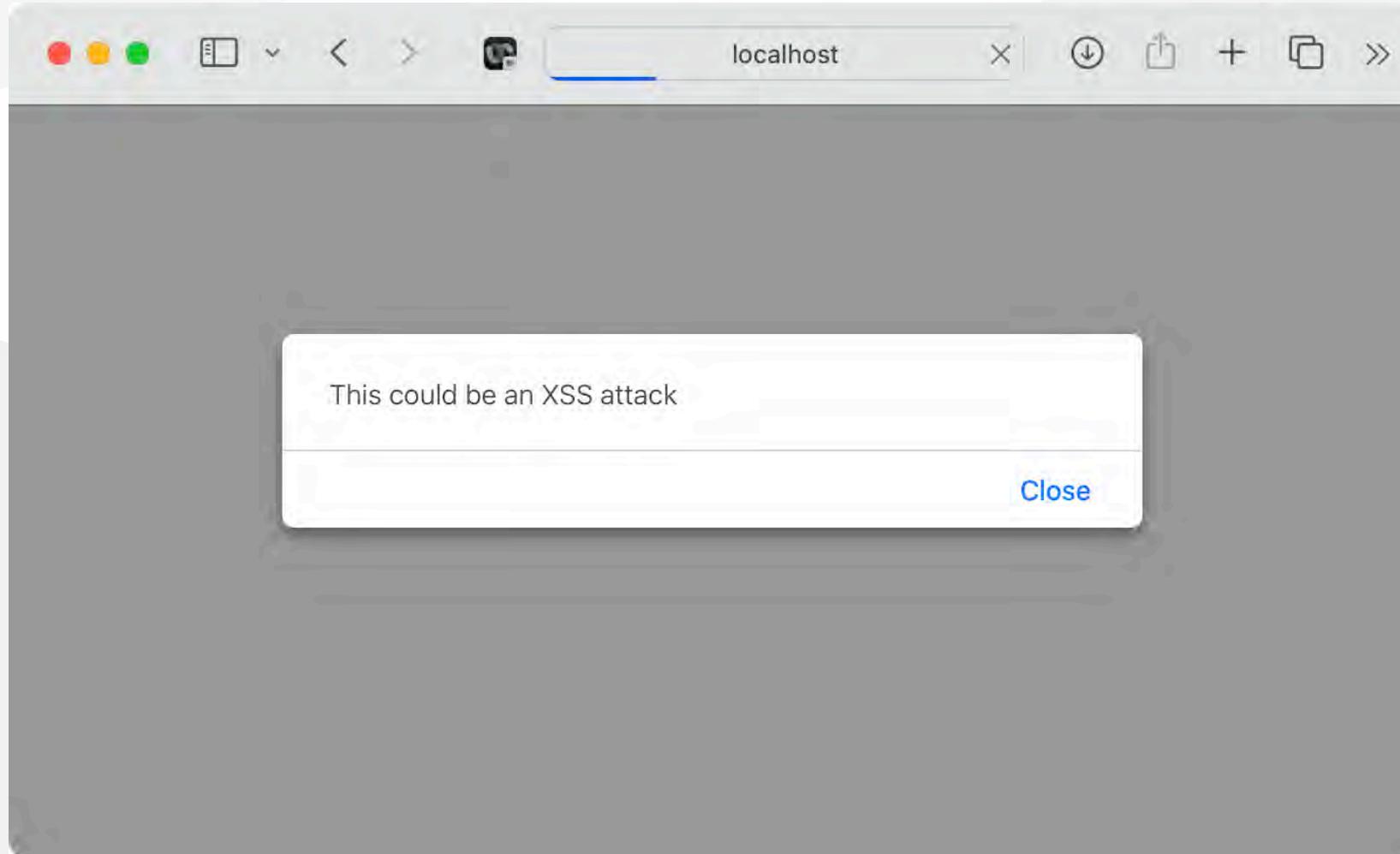
Ejemplo: riesgo de no escapar la salida

```
① <a class="badlink" href="xss-1.php?msg=<script>  
src=js/bad.js</script>">LINK TO DEMONSTRATE XSS</a>  
  
<?php  
② $message = $_GET['msg'] ?? 'Click link at top of page';  
?  
...  
<h1>XSS Example</h1>  
③ <p><?= $message ?></p>
```

Ejemplo: riesgo de no escapar la salida



Ejemplo: riesgo de no escapar la salida



Ejemplo: riesgo de no escapar la salida

¿Qué tipo de problemas podría originar un ataque XSS para los usuarios? ¿Y para el sitio web?

Escape de caracteres HTML reservados

La función integrada `htmlspecialchars()` de PHP sustituye los caracteres reservados de HTML por sus entidades correspondientes, de forma que dichos caracteres se muestren y no puedan ejecutarse como código.

Escape de caracteres HTML reservados

La función `htmlspecialchars()` tiene cuatro parámetros:

- `$text` es el texto que desea escapar.
- `$flag` es una opción para controlar qué caracteres se codifican (consultar la tabla siguiente para las opciones más comunes).

Escape de caracteres HTML reservados

- `$encoding` es el esquema de codificación utilizado en la cadena (si no se especifica, el valor por defecto es UTF-8).
- `$double_encode` Dado que las entidades HTML comienzan con un ampersand (`&`), si una cadena contiene una entidad, el ampersand se codifica y la página muestra la entidad (en lugar del carácter reservado). Utilizar un valor de `false` para este parámetro le dice al intérprete de PHP que no codifique entidades en la cadena.

```
htmlspecialchars($text[, $flag][, $encoding][, $double_encode]);
```

Escape de caracteres HTML reservados

Si la cadena que se está escapando está formada por caracteres que son todos válidos para el esquema de codificación que se utiliza, la función devuelve la cadena con los caracteres reservados sustituidos por entidades.

Si la cadena contiene caracteres no válidos, devuelve una cadena vacía (a menos que se utilice el indicador `ENT_ SUBSTITUTE`, como se describe en la tabla siguiente).

Escape de caracteres HTML reservados

FLAG	DESCRIPTION
<code>ENT_COMPAT</code>	Convert double quotes, leave single quotes alone (this is the default if no flag is supplied)
<code>ENT_QUOTES</code>	Convert double and single quotes
<code>ENT_NOQUOTES</code>	Do not convert double or single quotes
<code>ENT_SUBSTITUTE</code>	To prevent the function returning an empty string, replace invalid characters with the replacement character: <code>?</code> (in UTF-8 this is U+FFFD, in any other encoding it is <code>�</code>)
<code>ENT_HTML401</code>	Treat code as HTML 4.01
<code>ENT_HTML5</code>	Treat code as HTML 5
<code>ENT_XHTML</code>	Treat code as XHTML

To specify multiple flags, separate each one with a pipe symbol, e.g., `ENT_QUOTES|ENT_HTML5`

Escape de caracteres HTML reservados

Dado que `htmlspecialchars()` es un nombre de función bastante largo y tiene cuatro parámetros, algunos programadores crean funciones definidas por el usuario con un nombre más corto para escapar valores y devolver la versión codificada (como se muestra en la página de la derecha).

Ejemplo: escapando contenido proporcionado por usuarios

Vamos a ver dos ejemplos de escapado de caracteres mediante `htmlspecialchars()`, en el primero se utiliza dicha función directamente mientras que en el segundo se crear una función definida por el usuario `html_escape()` que utiliza internamente `htmlspecialchars()`:

Ejemplo: escapando contenido proporcionado por usuarios

1. El primer ejemplo sólo tiene un cambio con respecto al ejemplo anterior; cuando se escribe el valor en `$message`, utiliza la función `htmlspecialchars()` de PHP para reemplazar los caracteres reservados de HTML por sus entidades correspondientes. Por lo tanto, cuando se haga clic en el enlace, el HTML de las etiquetas `<script>` se mostrará en la pantalla, en lugar de ser ejecutado por el navegador.

Ejemplo: escapando contenido proporcionado por usuarios

2. Una segunda versión del mismo ejemplo añade una función definida por el usuario llamada `html_escape()`. Acepta una cadena como argumento, y devuelve esa cadena con todos los caracteres reservados sustituidos por entidades. Cuando llama a `htmlspecialchars()`, proporciona valores para los cuatro parámetros.
3. Finalmente se llama a la función `html_escape()` para escribir el mensaje de la cadena de consulta.

El resultado de ambos ejemplos es exactamente el mismo.

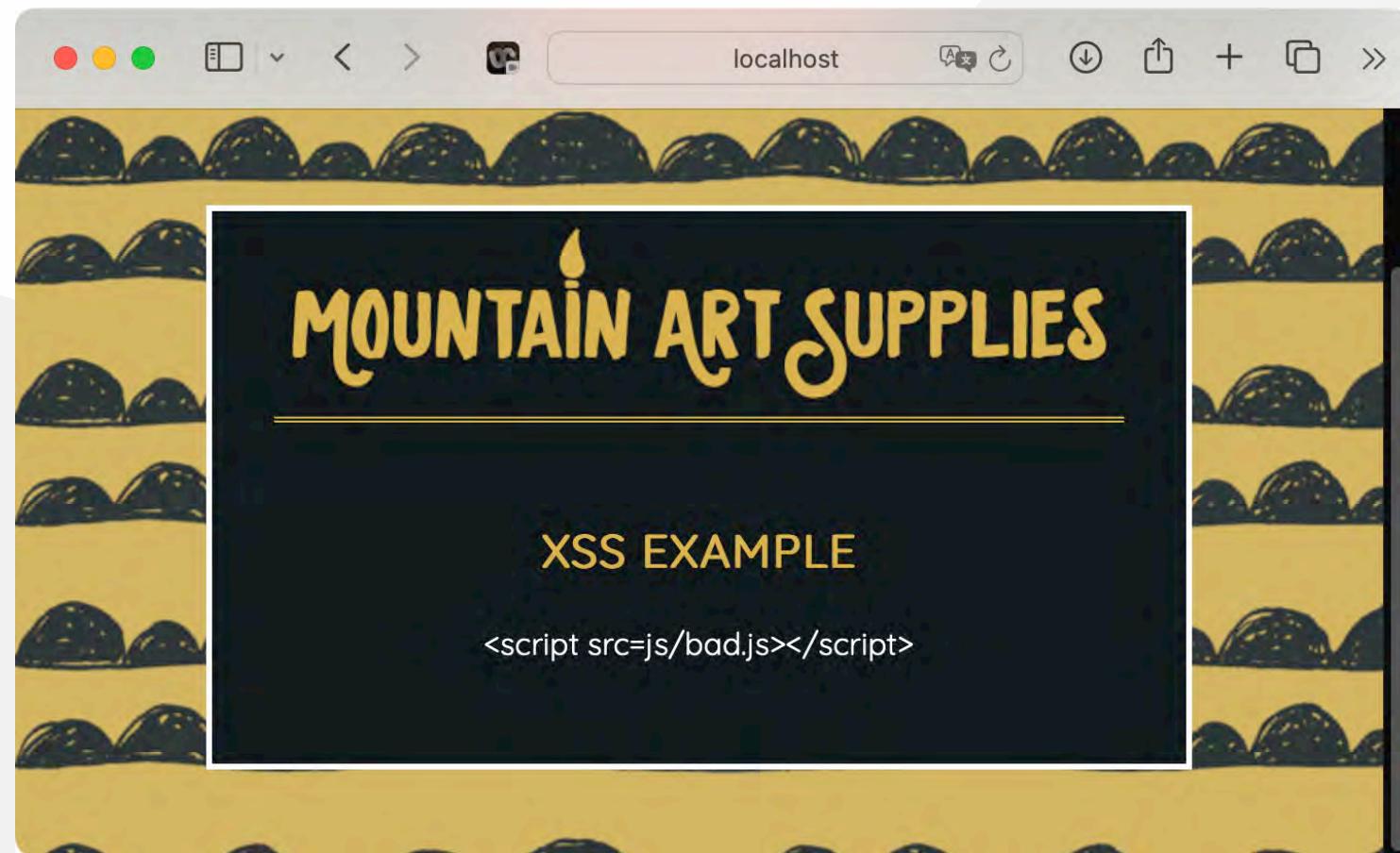
Ejemplo: escapando contenido proporcionado por usuarios

```
<a class="badlink" href="xss-2.php?msg=<script>  
src=js/bad.js</script>">ESCAPING MARKUP</a>  
  
<?php  
    $message = $_GET['msg'] ?? 'Click the link above';  
?> ...  
<h1>XSS Example</h1>  
① <p><?= htmlspecialchars($message) ?></p>
```

Ejemplo: escapando contenido proporcionado por usuarios

```
<a class="badlink" href="xss-3.php?msg=<script>  
src=js/bad.js</script>">ESCAPING MARKUP</a>  
  
<?php  
function html_escape(string $string): string  
{  
    ② - [ return htmlspecialchars($string,  
        ENT_QUOTES|ENT_HTML5, 'UTF-8', true);  
    ]  
}  
$message = $_GET['msg'] ?? 'Click the link above';  
?> ...  
<h1>XSS Example</h1>  
③ <p><?= html_escape($message) ?></p>
```

Ejemplo: escapando contenido proporcionado por usuarios



Ejemplo: escapando contenido proporcionado por usuarios

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

Vamos a realizar la siguiente tarea para practicar cómo escapar caracteres especiales en HTML utilizando la función `htmlspecialchars()` de PHP para prevenir ataques de Cross-Site Scripting (XSS).

Ejemplo: escapando contenido proporcionado por usuarios

Instrucciones:

- 1. Crear el archivo principal `index.php`**: Este archivo contendrá un enlace con un mensaje potencialmente malicioso que será enviado mediante la cadena de consulta.
- 2. Crear el archivo `xss.php` para procesar la cadena de consulta**: Este archivo recibirá el mensaje desde la cadena de consulta y lo mostrará en la página, asegurándose de escapar los caracteres especiales usando la función `htmlspecialchars()`.

Ejemplo: escapando contenido proporcionado por usuarios

3. Tareas a realizar:

- i. Ejecuta `index.php` en tu servidor web local.
- ii. Haz clic en el enlace del mensaje para ver cómo se maneja la entrada de usuario potencialmente peligrosa.
- iii. Modifica el enlace en `index.php` para probar con diferentes mensajes y observe cómo `htmlspecialchars()` escapa los caracteres especiales.
- iv. Experimenta con diferentes opciones de la función `htmlspecialchars()`, como `ENT_NOQUOTES`, `ENT_COMPAT`, etc., y muestra cómo cambian los resultados.

Cómo se envían los datos del formulario al servidor

Los **formularios** permiten a los visitantes introducir texto y seleccionar opciones. Para cada control de formulario, el navegador puede enviar un nombre y un valor al servidor junto con una solicitud de página.

Cómo se envían los datos del formulario al servidor

La etiqueta HTML `<form>` requiere dos atributos:

- El valor del atributo *action* es el archivo PHP al que deben enviarse los datos del formulario.
- El valor del atributo *method* indica cómo enviar los datos del formulario al servidor.

El atributo *method* debe tener uno de dos valores:

- *GET* envía los datos del formulario utilizando HTTP GET en una cadena de consulta añadida al final de la URL.
- *POST* envía los datos utilizando HTTP POST en las cabeceras HTTP enviadas desde el navegador al servidor.

Cómo se envían los datos del formulario al servidor

```
PAGE TO SEND DATA TO  
HTTP METHOD TO SEND DATA  
<form action="join.php" method="POST">  
  <p>Email: <input type="email" name="email"></p>  
  <p>Age: <input type="number" name="age"></p>  
  <p><input type="checkbox" name="terms" value="true">  
    I agree to the terms and conditions.</p>  
  <input type="submit" value="Save">  
</form>
```

Cómo se envían los datos del formulario al servidor

Cuando el visitante envía el formulario, el navegador solicita la página especificada en el atributo `action`.

El valor del atributo `action` puede ser una ruta relativa desde la página que crea el formulario hasta la página que procesa el formulario, o puede ser una URL completa.

A menudo, el formulario se enviará a la misma página PHP que se utilizó para mostrar el formulario.

Cómo se envían los datos del formulario al servidor

El formulario anterior se envía mediante HTTP POST, por lo que el navegador añadirá los nombres y valores de los controles del formulario a las cabeceras HTTP. Las cabeceras se envían con la solicitud de join.php. Para cada cabecera:

- El **nombre** es el valor del atributo name de ese control de formulario.
- El **valor** es el texto que el usuario introdujo o el valor del elemento que seleccionó.

Cómo se envían los datos del formulario al servidor

Los controles de formulario HTML que se muestran a continuación pertenecen a una de estas dos categorías: **entradas de texto**, que permiten a los visitantes introducir texto, y **opciones**, que permiten a los visitantes seleccionar una opción.

Cómo se envían los datos del formulario al servidor

Si un visitante rellena una **entrada de texto**, el nombre que se envía al servidor es el valor del atributo *name*, y el valor es el texto que ha introducido.

- Si el usuario no introduce ningún texto para ese control de formulario, el nombre sigue enviando al servidor y el valor es una cadena en blanco.

Si se selecciona una **opción**, el nombre es el valor del atributo *name* y el valor son los datos del atributo *value* para la opción que seleccionaron.

- Si el usuario no seleccionó una opción, el navegador no envía ningún dato para ese control de formulario al servidor.

Cómo se envían los datos del formulario al servidor

TEXT INPUT	EXAMPLE	PURPOSE
Text input	<code><input type="text" name="username"></code>	Enter single line of text
Number input	<code><input type="number" name="age"></code>	Enter number
Email input	<code><input type="email" name="email"></code>	Enter email
Password	<code><input type="password" name="password"></code>	Enter password
Text area	<code><textarea name="bio"></textarea></code>	Enter longer text

OPTION	EXAMPLE	PURPOSE
Radio buttons	<code><input type="radio" name="rating" value="good"></code> <code><input type="radio" name="rating" value="bad"></code>	Select one of multiple options
Select boxes	<code><select name="preferences"></code> <code> <option value="email">Email</option></code> <code> <option value="phone">Phone</option></code> <code></select></code>	Select one of multiple options
Checkboxes	<code><input type="checkbox" name="terms" value="true"></code>	Select a single option

Cómo se envían los datos del formulario al servidor

Para demostrar la validación del lado del servidor, en estos ejemplos sólo se valida los datos en el servidor. Los sitios web reales puestos en producción deben utilizar JavaScript para validar los datos en el navegador antes de enviarlos al servidor, y luego validar los datos de nuevo en el servidor (porque es posible eludir la validación en el navegador).

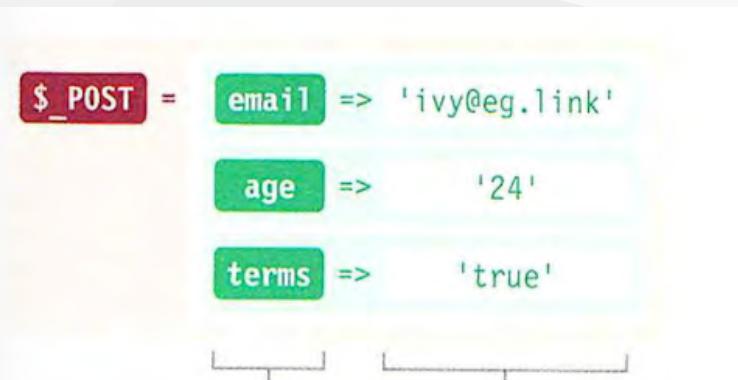
NOTA: Cuando el intérprete PHP añade datos desde el navegador a un array superglobal, siempre es un tipo de datos *string*, incluso si el valor es un número o un booleano. Conoceremos el control de subida de archivos utilizado para enviar archivos al servidor en la próxima unidad.

Obtener datos de un formulario

Cuando el intérprete de PHP recibe datos enviados a través de HTTP POST, se añaden al array superglobal `$_POST`.

Cuando el visitante envía un formulario a través de HTTP POST, el intérprete de PHP recibe la petición de la página y añade los datos del formulario (enviados en las cabeceras HTTP) al array superglobal `$_POST`:

- La **clave (key)** es el nombre del control del formulario.
- El **valor (value)** es el valor que el usuario ha introducido o seleccionado.



Obtener datos de un formulario

Si el formulario se envió utilizando HTTP GET, el intérprete de PHP obtiene los datos del formulario de la cadena de consulta y los añade al array superglobal `$_GET`.

El código del fichero PHP puede acceder a los valores del array superglobal `$_POST` de la misma forma que accedería a los valores de cualquier array asociativo.

Si el control del formulario es una entrada de texto, siempre habrá un valor para él (a menos que haya sido desactivado):



Obtener datos de un formulario

Si el control del formulario es una opción, el nombre y el valor sólo se añaden a las cabeceras HTTP si el visitante selecciona una opción. Por lo tanto, se utiliza el operador de coalescencia nula (??) para recoger opciones del array superglobal `$_POST` (del mismo modo que se utilizó para recoger valores de la cadena de consulta).

```
$age = $_POST['age'] ?? false;  
      ^          ^  
      VARIABLE    KEY      DEFAULT VALUE
```

Ejemplo: Cómo se reciben los datos de los formularios

El siguiente ejemplo muestra lo que contienen los arrays superglobales cuando las páginas utilizan formularios.

La función `var_dump()` (que ya vimos en la introducción de este bloque) se utiliza para mostrar el contenido del array superglobal, de modo que podamos ver qué elementos se añaden al array, y también para que se pueda ver que todos los datos de estos arrays superglobal son de tipo cadena (*string*), aunque sean números o booleanos.

Ejemplo: Cómo se reciben los datos de los formularios

Es importante que pruebes este ejemplo por ti mismo y veas cómo cambian los datos en el array superglobal cuando:

- La página se carga por primera vez, antes de enviar el formulario
- Se envía el formulario, sin llenar ningún dato
- Se llenan los campos del formulario

Ejemplo: Cómo se reciben los datos de los formularios

1. Cinco controles de texto piden el nombre, la edad, el correo electrónico, la contraseña y la biografía del usuario.
2. Tres controles de formulario presentan opciones al visitante.
3. El contenido del array superglobal `$_POST` se escribe utilizando la función
`var_dump()`.

Ejemplo: Cómo se reciben los datos de los formularios

Cuando la página se carga, el formulario no ha sido enviado, por lo que el superglobal `$_POST` estará vacío.

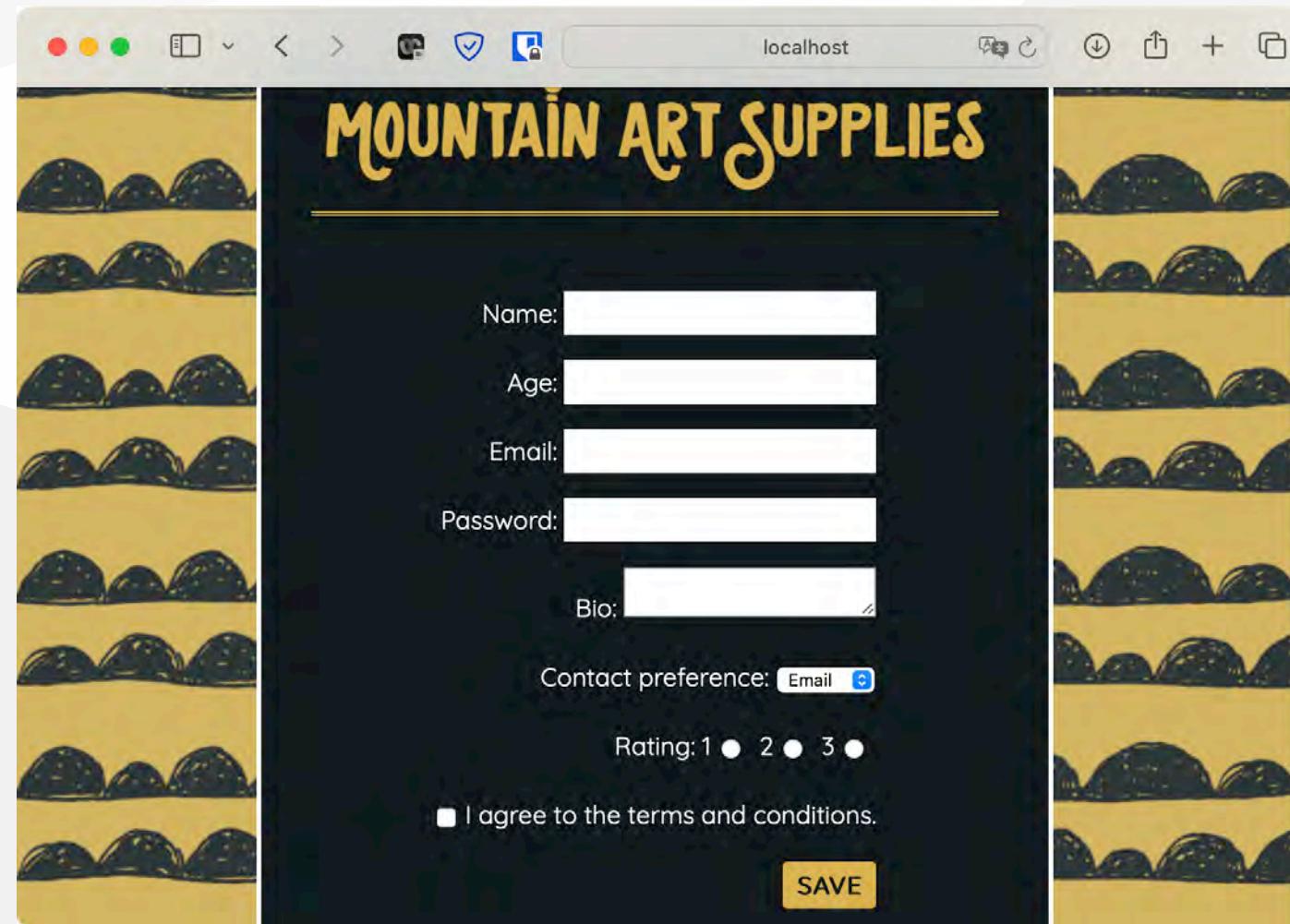
Si el formulario se envía sin introducir ningún dato, la matriz superglobal `$_POST` contiene un elemento para cada una de las entradas de texto; su valor es una cadena en blanco. La caja de selección se envía al servidor con el valor por defecto mostrado al cargar la página. Pero los nombres y valores de los botones de radio y la casilla de verificación no se envían al servidor.

Si se rellenan todos los controles del formulario, el array superglobal `$_POST` contendrá un elemento por cada control del formulario. Cada valor enviado al servidor es una cadena.

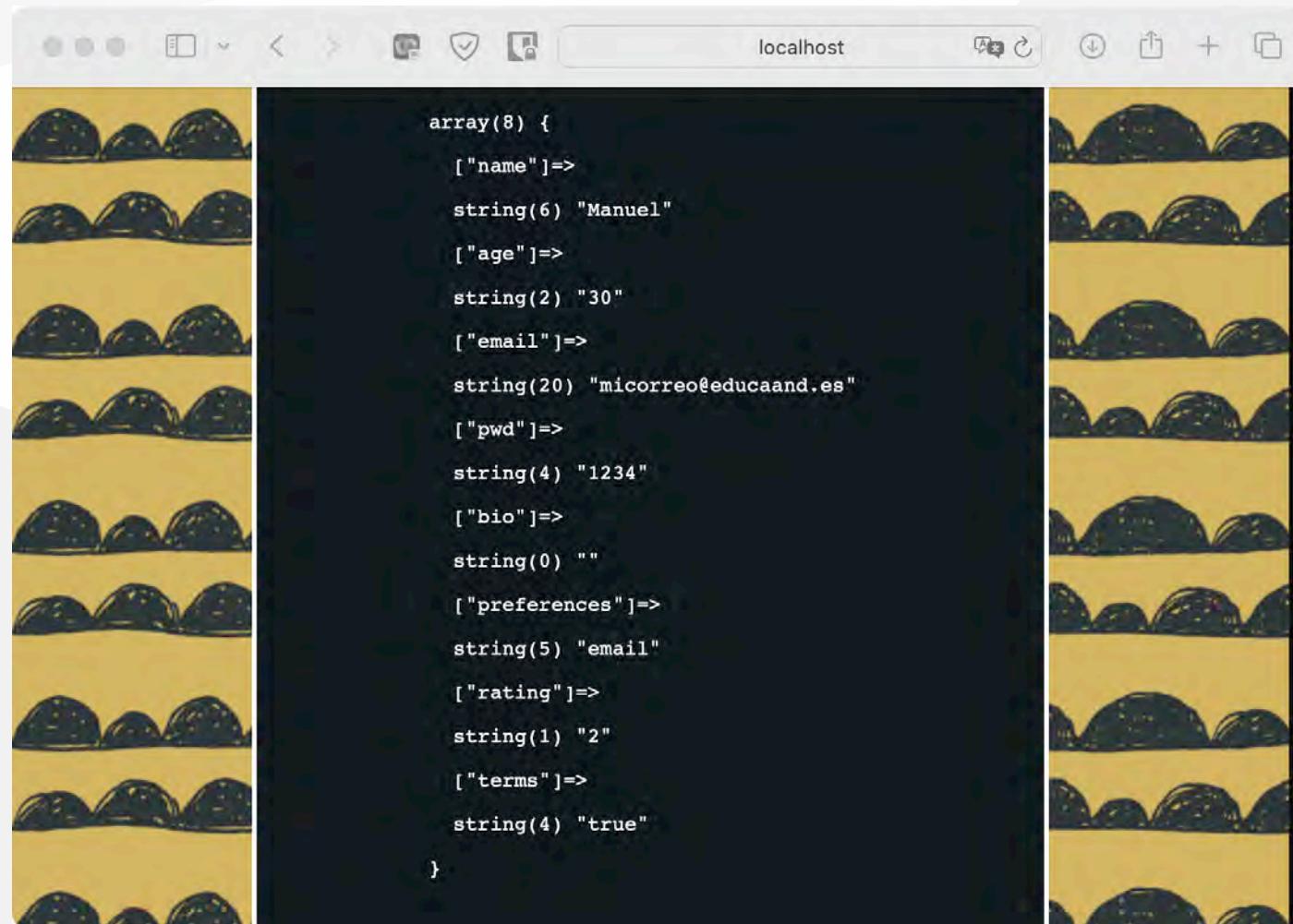
Ejemplo: Cómo se reciben los datos de los formularios

```
<form action="collecting-form-data.php" method="POST">
    <p>Name:      <input type="text" name="name"></p>
    <p>Age:       <input type="text" name="age"></p>
    <p>Email:      <input type="text" name="email"></p>
    <p>Password:   <input type="password" name="pwd"></p>
    <p>Bio:        <textarea name="bio"></textarea></p>
    <p>Contact preference:
        <select name="preferences">
            <option value="email">Email</option>
            <option value="phone">Phone</option>
        </select></p>
    <p>Rating:
        1 <input type="radio" name="rating" value="1">&ampnbsp;
        2 <input type="radio" name="rating" value="2">&ampnbsp;
        3 <input type="radio" name="rating" value="3"></p>
    <p><input type="checkbox" name="terms" value="true">
        I agree to the terms and conditions.</p>
    <p><input type="submit" value="Save"></p>
</form>
③ <pre><?php var_dump($_POST); ?></pre>
```

Ejemplo: Cómo se reciben los datos de los formularios



Ejemplo: Cómo se reciben los datos de los formularios



A screenshot of a web browser window titled "localhost". The page content is a JSON object displayed in a dark-themed code editor. The JSON structure is as follows:

```
array(8) {  
    [ "name" ]=>  
        string(6) "Manuel"  
    [ "age" ]=>  
        string(2) "30"  
    [ "email" ]=>  
        string(20) "micorreo@educaand.es"  
    [ "pwd" ]=>  
        string(4) "1234"  
    [ "bio" ]=>  
        string(0) ""  
    [ "preferences" ]=>  
        string(5) "email"  
    [ "rating" ]=>  
        string(1) "2"  
    [ "terms" ]=>  
        string(4) "true"  
}
```

Ejemplo: Cómo se reciben los datos de los formularios

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- Cambia el valor del atributo method de la etiqueta `<form>` a GET, y los datos se enviarán a través de HTTP GET. A continuación, en el paso 3, muestra el contenido del array superglobal `$_GET`.



6. Comprobación y Validación de Formularios

Comprobar que se ha enviado un formulario

Un formulario debe ser enviado antes de poder recoger y procesar sus datos. Existen diferentes técnicas para comprobar si el formulario se ha enviado dependiendo de si el formulario se envió a través de HTTP POST o HTTP GET.

Comprobar que se ha enviado un formulario

HTTP POST

El array superglobal `$_SERVER` (visto en la introducción al bloque B) tiene una clave llamada `REQUEST_METHOD`, que almacena el método HTTP utilizado para solicitar la página. Cuando un formulario se envía utilizando HTTP POST, tiene un valor de POST.

Para comprobar si un formulario se ha enviado utilizando HTTP POST, la condición de una sentencia `if` comprueba si la clave `REQUEST_METHOD` tiene el valor POST. El código para procesar el formulario se incluye en el siguiente bloque de código.

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {  
    // Code to collect and process form data goes here  
}
```

Comprobar que se ha enviado un formulario

HTTP GET

Cuando un usuario hace clic en un enlace o introduce una URL en la barra de direcciones del navegador, la solicitud siempre se envía a través de HTTP GET. Por lo tanto, no se puede utilizar el superglobal `$_SERVER` para comprobar si un formulario se ha enviado a través de HTTP GET. En su lugar, puedes añadir:

- Una entrada oculta al formulario, o
- Nombre y valor al botón de envío

Comprobar que se ha enviado un formulario

HTTP GET

Cuando se envíe el formulario, el nombre y el valor de la entrada oculta o del botón de envío se añadirán al array superglobal `$_GET`.

La condición de una sentencia if puede comprobar si el superglobal `$_GET` tiene el valor que se envía cuando se envía el formulario. Si lo tiene, el código para recoger y procesar los datos podrá ejecutarse.

```
$submitted = $_GET['submitted'] ?? '';
if ($submitted === 'true') {
    // Code to collect and process form data goes here
}
```

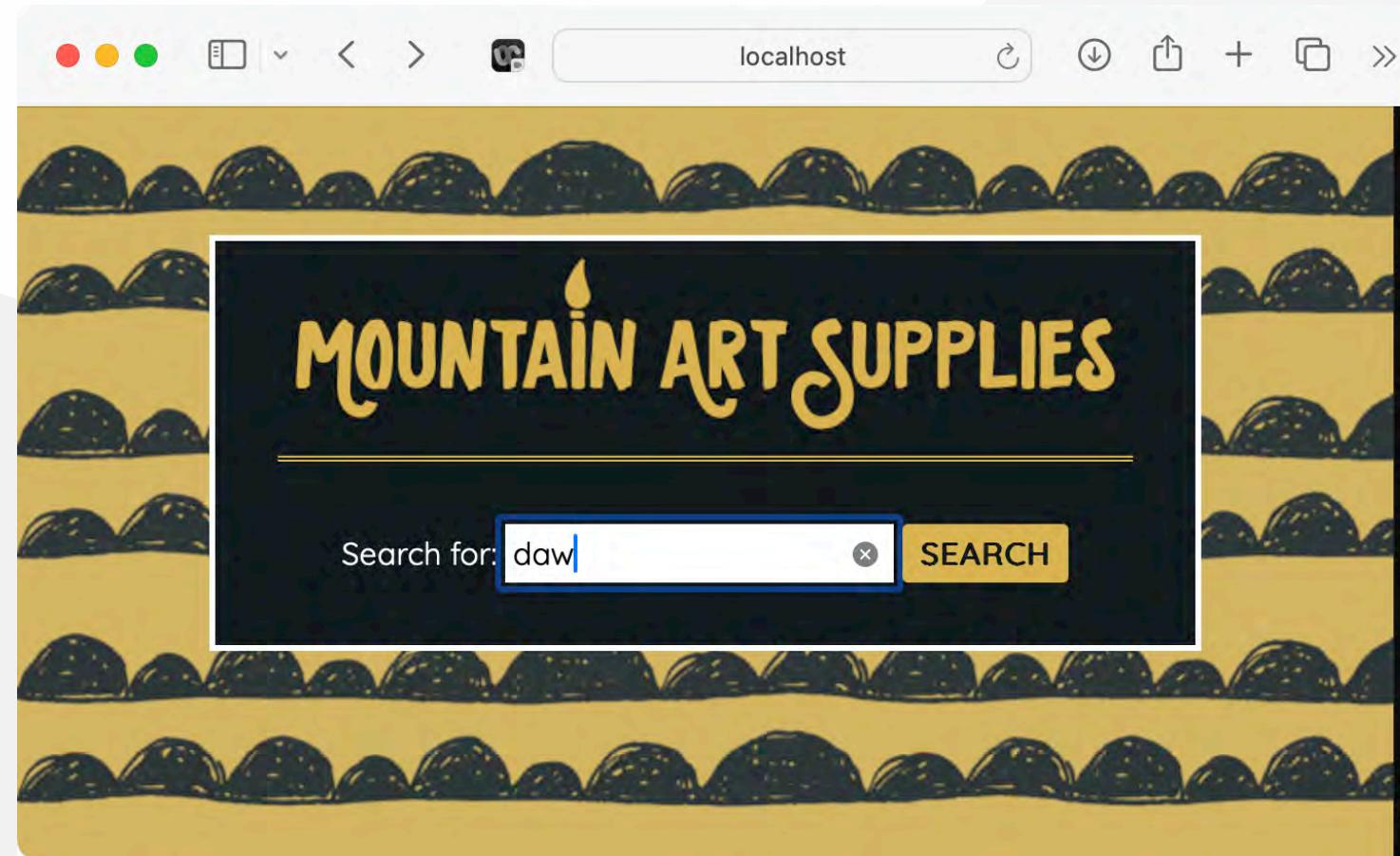
Ejemplo: Comprobar que se ha enviado un formulario (HTTP POST)

1. La condición de una sentencia `if` comprueba el array superglobal `$_SERVER` para ver si la clave llamada `REQUEST_METHOD` tiene el valor POST.
2. Si lo tiene, el formulario de búsqueda se ha enviado a través de HTTP POST, y se utilizará un mensaje para mostrar el término de búsqueda.
3. En caso contrario, se omite y se muestra el formulario.

Ejemplo: Comprobar que se ha enviado un formulario (HTTP POST)

```
<?php  
① if ($_SERVER['REQUEST_METHOD'] == 'POST') {  
    ② }  
    $term = $_POST['term'];  
    echo 'You searched for ' . htmlspecialchars($term);  
} else { ?>  
    <form action="check-for-http-post.php" method="POST">  
        Search for: <input type="text" name="term">  
        <input type="submit" value="search">  
    </form>  
<?php } ?>
```

Ejemplo: Comprobar que se ha enviado un formulario (HTTP POST)



Ejemplo: Comprobar que se ha enviado un formulario (HTTP GET)

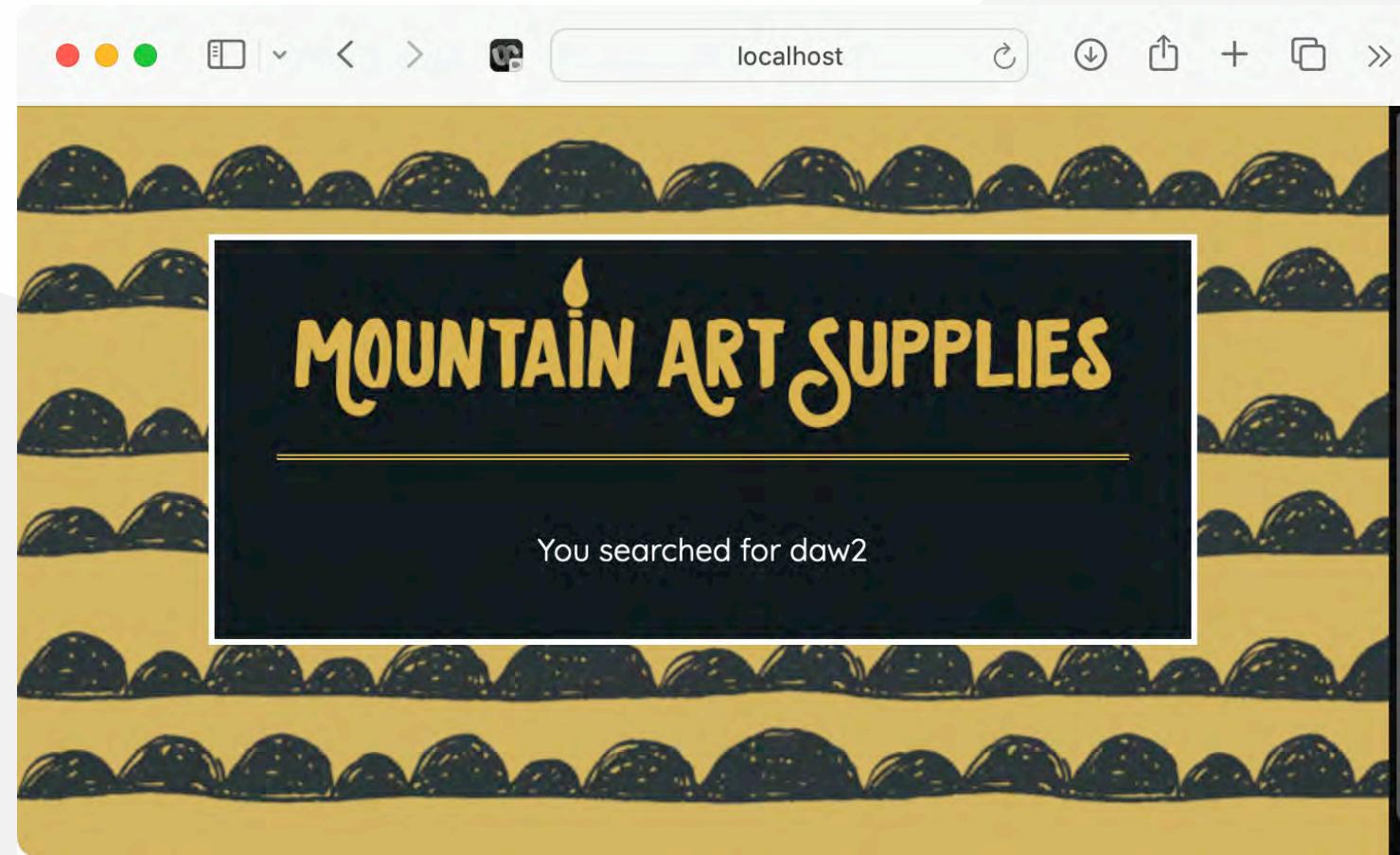
En este ejemplo, se envía el nombre del botón de envío y su valor es búsqueda. Si el formulario fue enviado, se añaden al array superglobal `$_GET`.

4. El operador de coalescencia nula comprueba si el array superglobal `$_GET` tiene un valor para la clave enviada. Si lo tiene, una variable llamada `$submitted` almacena su valor; si no lo tiene, almacena una cadena en blanco.
5. La condición de una sentencia `if` comprueba si el valor en `$submitted` es `search`. Si lo es, el formulario se envió a través de HTTP GET y se muestra el término de búsqueda.
6. En caso contrario, se muestra el formulario.

Ejemplo: Comprobar que se ha enviado un formulario (HTTP GET)

```
<?php  
④ $submitted = $_GET['sent'] ?? '';  
⑤ if ($submitted === 'search') {  
    $term = $_GET['term'] ?? '';  
    echo 'You searched for ' . htmlspecialchars($term);  
} else { ?>  
    ⑥ <form action="check-for-http-get.php" method="GET">  
        Search for: <input type="search" name="term">  
        <input type="submit" name="sent" value="search">  
    </form>  
<?php } ?>
```

Ejemplo: Comprobar que se ha enviado un formulario (HTTP GET)



Ejemplo: Comprobar que se ha enviado un formulario

En esta actividad vamos a practicar cómo enviar y procesar datos utilizando formularios HTML en PHP, verificando si el formulario ha sido enviado previamente.

Ejemplo: Comprobar que se ha enviado un formulario

Instrucciones:

1. Crear el archivo principal `registro.php` :

Este archivo contendrá el formulario de registro para un club de fútbol. El formulario debe solicitar el nombre, apellido, edad y posición del jugador. Además, debe manejar el envío del formulario y mostrar un mensaje de confirmación.

2. Crear archivos de cabecera y pie de página `header.php` y `footer.php` :

Estos archivos se incluirán en el archivo principal para estructurar la página de manera adecuada.

Ejemplo: Comprobar que se ha enviado un formulario

3. Tareas a realizar:

- i. Ejecuta `registro.php` en tu servidor web local.
- ii. Completa el formulario de registro y envíalo.
- iii. Observa cómo los datos se procesan y se muestran en la misma página después de enviar el formulario.
- iv. Modifica el formulario para agregar más campos, como la dirección de correo electrónico o el número de teléfono, y asegúrate de que se procesen correctamente.
- v. Modifica el formulario para enviar los datos utilizando el método `GET` y ajuste el procesamiento de datos en consecuencia.



7. Validación Específica de Datos: Números y Texto

Validando números

Cuando se recogen los datos del formulario, deben validarse para garantizar que se han proporcionado todos los valores requeridos y que los datos están en el formato correcto. Así se evita que los datos erróneos provoquen errores al ejecutar la página.

Validando números

Para comprobar si un valor es un número, utiliza la función integrada `is_numeric()` de PHP. O, si necesitas comprobar que un número está dentro de un rango especificado de números permitidos, puedes crear una función definida por ti mismo para realizar la tarea.

A continuación, se muestra una función que utiliza operadores de comparación para comprobar si un número se encuentra dentro del rango mínimo y máximo de valores permitidos. La función tiene tres parámetros:

- `$number` es el valor que necesita comprobar
- `$min` es el número mínimo permitido
- `$max` es el número máximo permitido

Validando números

```
function is_number($number, int $min = 0, int $max = 100): bool
{
    return ($number >= $min and $number <= $max);
}
    IS IT >= MINIMUM?           IS IT <= MAXIMUM?
```

Validando números

En la función, la condición contiene dos expresiones que comprueban si el número es:

- Mayor o igual (`>=`) que el número mínimo
- Menor o igual (`<=`) que el número máximo

Si ambas expresiones se evalúan como verdadero, la función devuelve verdadero. Si cualquiera de las expresiones resulta falsa, la función devuelve false.

Una vez que una página ha recogido un número, puede comprobar si el valor es válido llamando a esta función.

Validando números

Si los datos del formulario no son válidos, a menudo se vuelve a mostrar el formulario al usuario para que lo intente de nuevo. En tales casos, el número que proporcionaron puede mostrarse en el control del formulario escribiéndolo en el atributo value de la etiqueta

`<input>` .

Importante: Se utiliza la función `htmlspecialchars()` al mostrar el valor para evitar un ataque XSS.

Validando números

Dado que el valor introducido por el usuario sólo se recoge si el formulario ha sido enviado, la variable \$age debe declararse en la parte superior de la página y se le debe dar un valor inicial de una cadena en blanco.

Si la variable no se declarara en la parte superior de la página, al intentar mostrarla en el atributo value del control del formulario se produciría un error de variable no definida en la entrada de texto.

```
<input type="text" name="age" value="= htmlspecialchars($age) ?&gt;"&gt;</pre
```

Ejemplo: comprobando si un número es válido

En el siguiente ejemplo se comprueba que la edad introducida se encuentra entre los valores esperados (16-65). En caso contrario se muestra un mensaje de error:

1. Dos variables, `$age` y `$message`, se inicializan con valores que son cadenas en blanco.
2. Se define la función `is_number()` (la hemos explicado previamente).
3. La página comprueba si el formulario ha sido enviado. En caso afirmativo...

Ejemplo: comprobando si un número es válido

4. La edad se recoge del array superglobal `$_POST`. Los datos provienen de una entrada de texto, por lo que siempre se enviará un valor para ella cuando se envíe este formulario.
5. Se llama a la función `is_number()`. El valor enviado por el usuario es el primer argumento, y los números 16 y 65 son los números mínimo y máximo válidos. El valor booleano que devuelve se almacena en `$valid`.
6. La condición de una sentencia `if` comprueba si el valor en `$valid` es verdadero. Si lo es, la variable `$message` contiene un mensaje diciendo que la edad es válida.

Ejemplo: comprobando si un número es válido

7. En caso contrario, `$message` almacena un mensaje de error.
8. Se muestra el mensaje.
9. El número introducido por el usuario (o el valor inicial del paso 1) se muestra en la entrada de número utilizando `htmlspecialchars()`.

Ejemplo: comprobando si un número es valido

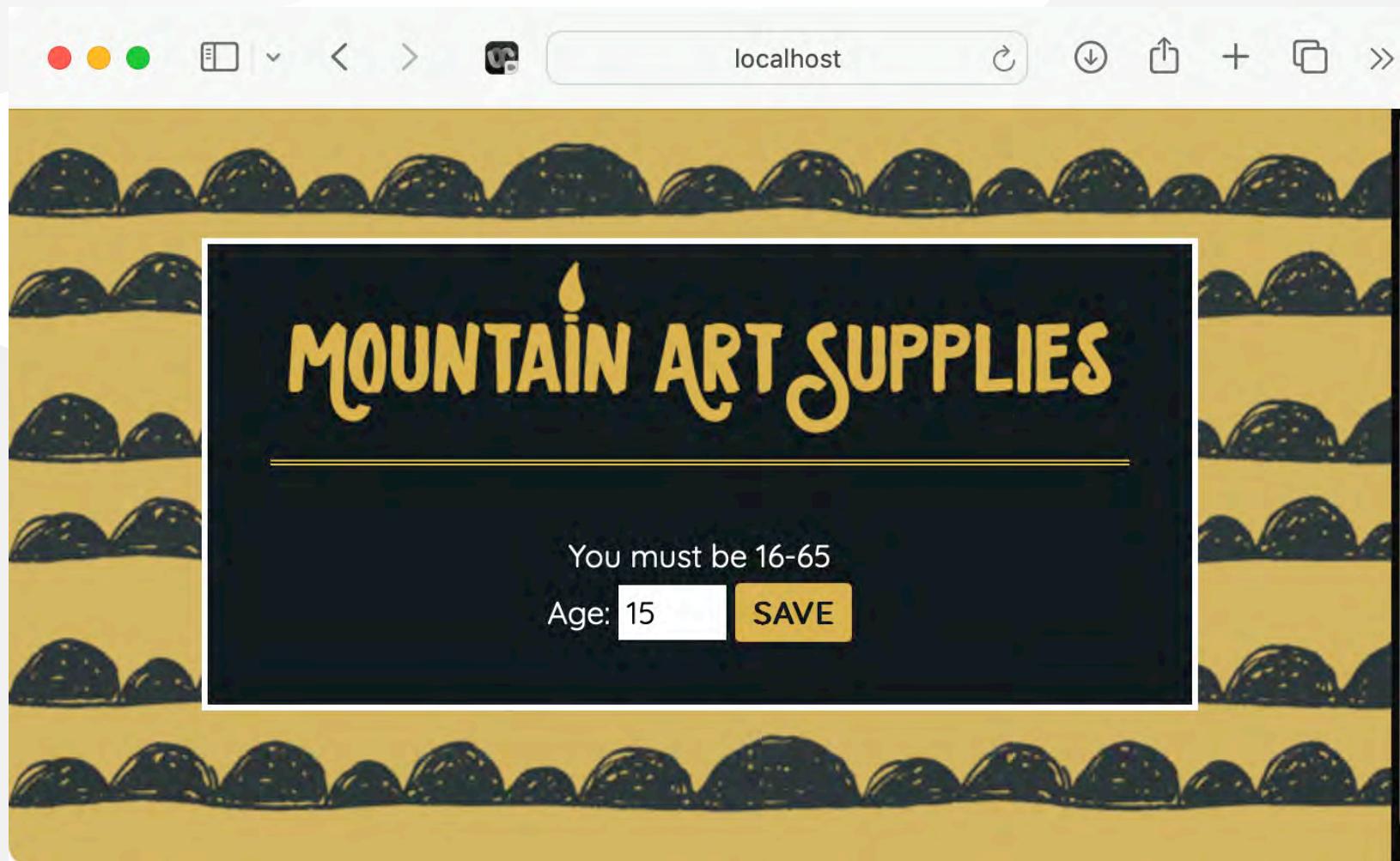
```
<?php
declare(strict_types = 1);

① $age      = '';
② $message = '';

function is_number($number, int $min = 0, int $max = 100): bool
{
    return ($number >= $min and $number <= $max);
}

③ if ($_SERVER['REQUEST_METHOD'] == 'POST') {
④     $age    = $_POST['age'];
⑤     $valid = is_number($age, 16, 65);
⑥     if ($valid) {
⑦         $message = 'Age is valid';
⑧     } else {
⑨         $message = 'You must be 16-65';
⑩     }
⑪     ?> ...
⑫     <?= $message ?>
⑬     <form action="validate-number-range.php" method="POST">
⑭         Age: <input type="text" name="age" size="4"
⑮             value="<?= htmlspecialchars($age) ?>">
⑯         <input type="submit" value="Save">
⑰     </form>
```

Ejemplo: comprobando si un número es valido



Ejemplo: comprobando si un número es valido

Tomando como punto de partida la actividad anterior (ver [Ejemplo: Comprobar que se ha enviado un formulario](#)), verifica que la edad introducida se encuentre en el rango [8-16]. En caso contrario muestra un error y vuelve a solicitar el formulario.

Validando la longitud de un texto

Los sitios web suelen limitar el número de caracteres que pueden aparecer en elementos como nombres de usuario, entradas, títulos de artículos y perfiles.

Se puede utilizar una única función para comprobar la longitud de cualquier cadena que reciba el sitio.

Validando la longitud de un texto

Para comprobar si el texto suministrado por los usuarios está entre un número mínimo y máximo de caracteres:

- Se utiliza la función incorporada de PHP `mb_strlen()` para contar cuántos caracteres hay en la cadena. Este número se almacena en una variable.
- A continuación, una condición utiliza dos expresiones para comprobar si el número de caracteres está **dentro del rango permitido** (del mismo modo que se utilizaron en el ejemplo anterior para comprobar que un número estaba dentro de un rango permitido).

Si el número de caracteres es válido, la función devuelve verdadero; en caso contrario, devuelve falso.

Validando la longitud de un texto

```
function is_text($text, int $min = 0, int $max = 100): bool
{
    $length = mb_strlen($text);
    return ($length >= $min and $length <= $max);
}
    IS IT >= MINIMUM?           IS IT <= MAXIMUM?
```

Validando la longitud de un texto

Cuando el código para validar datos se coloca en una función, puede utilizarse para validar múltiples controles de formulario. Esto ahorra repetir código para realizar la misma tarea.

La función anterior utiliza parámetros para que, cada vez que sean llamados, puedan tener diferentes valores mínimos y máximos.

Validando la longitud de un texto

Cuando varias páginas realizan las mismas tareas de validación, se debe colocar las definiciones de función en un archivo de inclusión.

Así podrás incluir ese archivo en lugar de duplicar las mismas definiciones de función en cada página.

El código de adjunto para esta unidad tiene un archivo de inclusión llamado `validate.php` que contiene tres definiciones de funciones que utilizaremos a lo largo de la unidad.

Ejemplo: Comprobando la longitud de un texto

En el siguiente ejemplo se comprueba si una entrada del usuario enviada por formulario mediante HTTP POST es válida en base a la longitud de la cadena de caracteres que introduzca el usuario.

1. Se inicializan las variables `$username` y `$message`.
2. Se define la función definida por el usuario llamada `is_text()` (mostrada en las diapositivas previas).
3. La página comprueba si el formulario ha sido enviado. En caso afirmativo...

Ejemplo: Comprobando la longitud de un texto

4. El texto se recoge del array superglobal `$_POST`.
5. Se llama a la función `is_text()` para comprobar si el texto introducido por el usuario tiene entre 3 y 18 caracteres. El valor que devuelve se almacena en `$valid`.
6. La condición de una sentencia `if` comprueba si el valor de `$valid` es verdadero. Si lo es, `$message` contiene un mensaje diciendo que el nombre de usuario es válido.
7. En caso contrario, `$message` almacena un mensaje indicando al usuario que el nombre de usuario debe tener entre 3 y 18 caracteres.
8. Se muestra el valor de la variable `$message`.
9. El valor en `$username` se muestra en la entrada de texto. Este es el valor que el usuario envió, o la cadena en blanco utilizada para inicializar la variable en el Paso 1.

Ejemplo: Comprobando la longitud de un texto

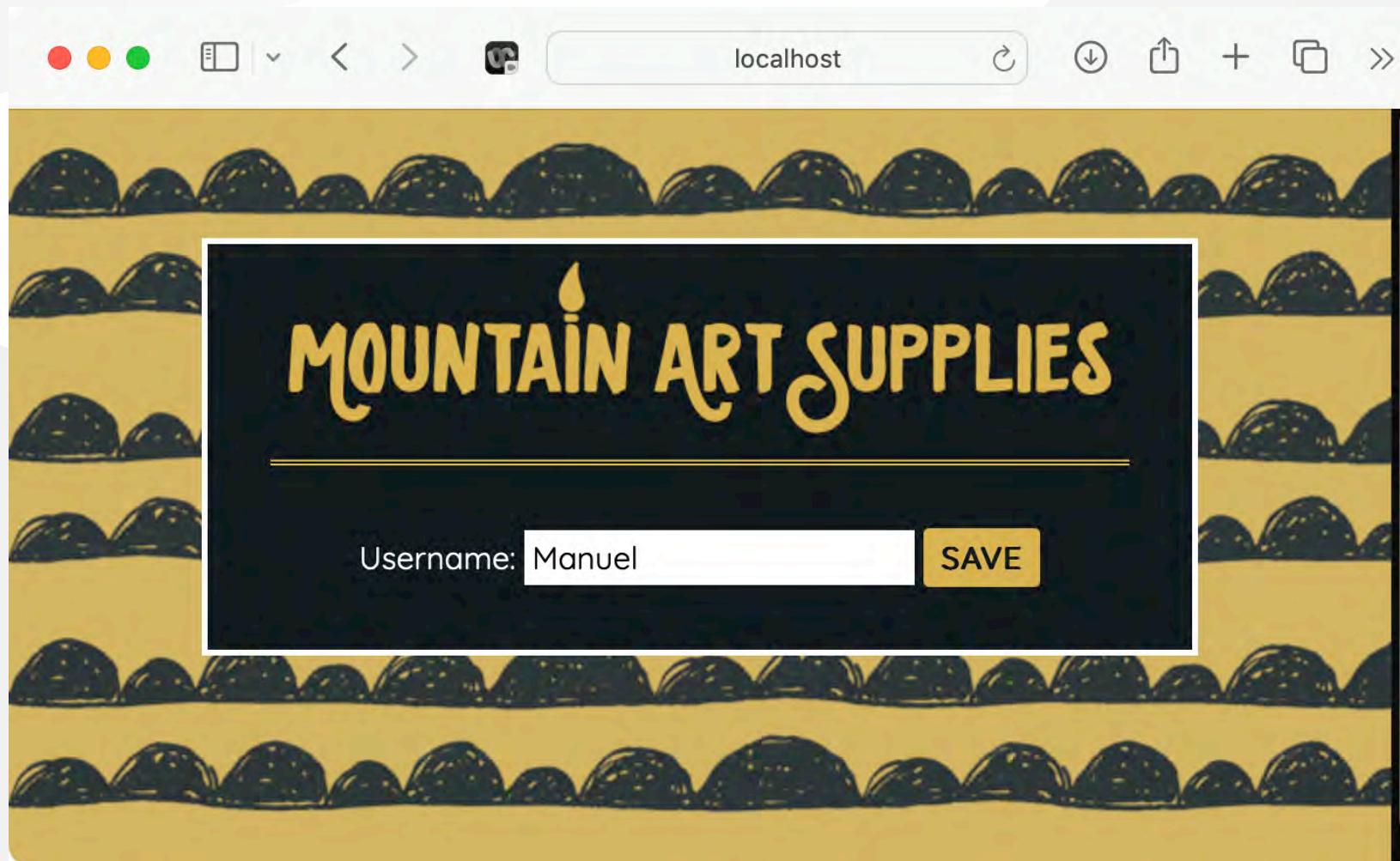
```
<?php
declare(strict_types = 1);
① $username = '';
① $message = '';

function is_text($text, int $min = 0, int $max = 1000): bool
{
    ② $length = mb_strlen($text);
    return ($length >= $min and $length <= $max);
}

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $username = $_POST['username'];
    $valid    = is_text($username, 3, 18);
    if ($valid) {
        $message = 'Username is valid';
    } else {
        $message = 'Username must be 3-18 characters';
    }
}
?> ...

<?= $message ?>
<form action="validate-text-length.php" method="POST">
    Username: <input type="text" name="username"
    ⑨     value="<?= htmlspecialchars($username) ?>">
        <input type="submit" value="Save">
</form>
```

Ejemplo: Comprobando la longitud de un texto



Validación de datos mediante expresiones regulares

Como vimos en la unidad anterior, las expresiones regulares pueden describir un patrón permitido de caracteres, como los utilizados en los números de tarjetas de crédito, códigos postales y números de teléfono.

La siguiente función utiliza expresiones regulares para comprobar la seguridad de las contraseñas de los usuarios.

Validación de datos mediante expresiones regulares

Acepta una contraseña como parámetro y comprueba si tiene 8 o más caracteres. A continuación, utiliza expresiones regulares para comprobar si contiene:

- Mayúsculas
- Minúsculas
- Números

Cada comprobación se separa mediante el operador *and*. Si todas las condiciones se evalúan como verdaderas, la función devuelve verdadero; en caso contrario, devuelve falso.

Podría utilizarse una única expresión regular para realizar todas las comprobaciones de una sola vez, pero la expresión regular sería más difícil de leer

Validación de datos mediante expresiones regulares

```
function is_password(string $password): bool
{
    if (
        mb_strlen($password) >= 8
        and preg_match('/[A-Z]/', $password)
        and preg_match('/[a-z]/', $password)
        and preg_match('/[0-9]/', $password)
    ) {
        return true; // Passed all tests
    }
    return false; // Invalid
}
```

Validación de datos mediante expresiones regulares

La función contiene una condición con cuatro expresiones:

- En primer lugar, `mb_strlen()` comprueba si el valor contiene 8 o más caracteres.
- A continuación, se utiliza tres veces la función `preg_match()` de PHP para comprobar si en la contraseña se encuentra el patrón de caracteres descrito en una expresión regular.
- Si todas las expresiones resultan en *true*, el siguiente bloque de código devuelve el valor *true* (porque cumple los requisitos).
- En caso contrario, si la función sigue en ejecución, devuelve *false*.

NOTA: Aunque los navegadores ocultan la contraseña cuando se teclea, los datos se siguen enviando como texto sin formato en las cabeceras HTTP. Por lo tanto, todos los datos personales deben enviarse a través de HTTPS.

Ejemplo: comprobar la seguridad de una contraseña

El siguiente código valida si la contraseña ingresada en un formulario cumple con ciertos requisitos de fortaleza, verificando que tenga al menos 8 caracteres, incluya al menos una letra mayúscula, una letra minúscula y un número, y muestra un mensaje indicando si la contraseña es válida o no.

1. Se inicializan las variables `$password` y `$message`.
2. Se define la función ^ con un parámetro: la contraseña a comprobar.
3. Una sentencia `if` utiliza cuatro expresiones; cada una da como resultado *verdadero* o *falso*. Están separadas por el operador `and`, de modo que el bloque de código subsiguiente sólo se ejecuta si todas resultan en *verdadero*.

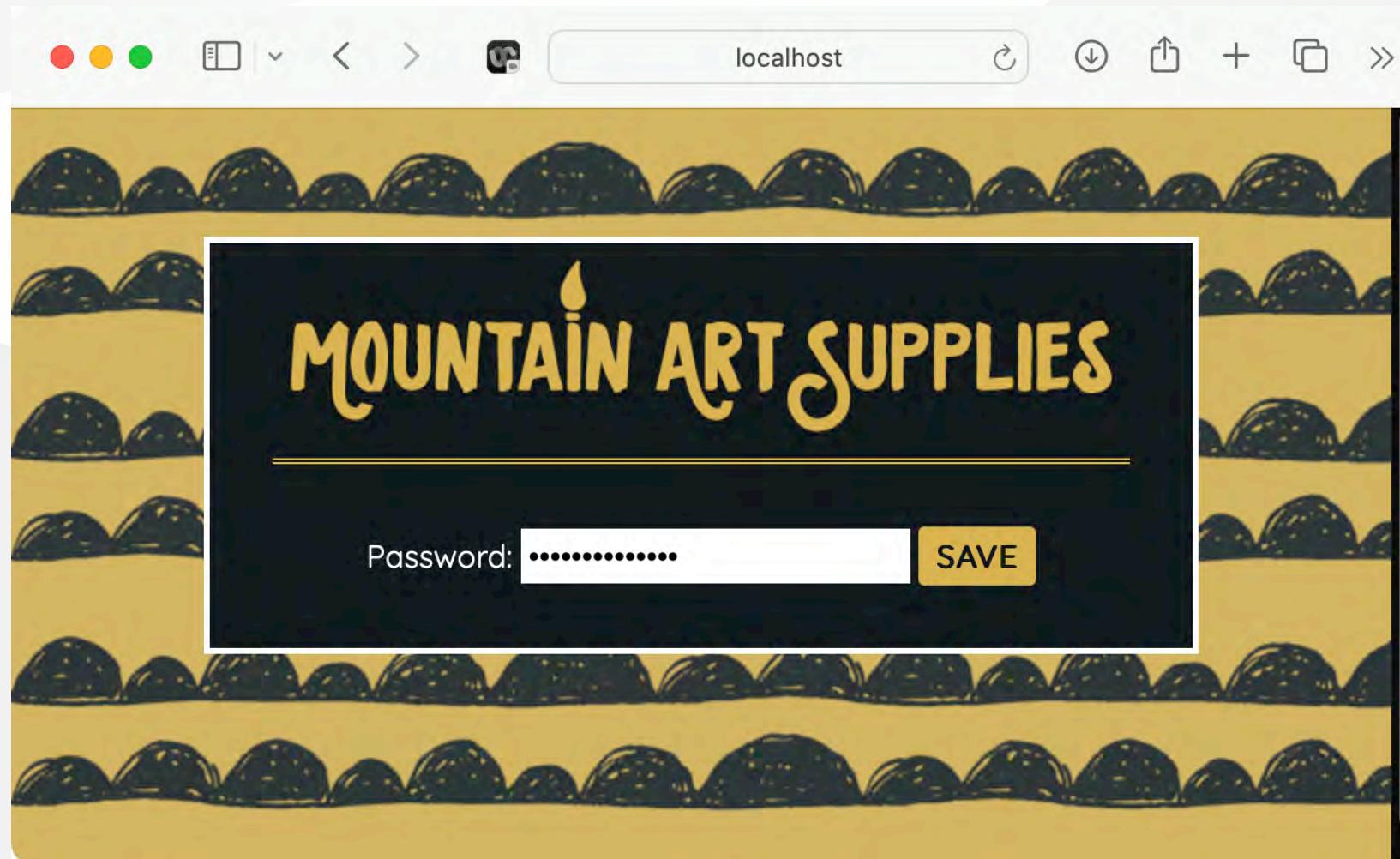
Ejemplo: comprobar la seguridad de una contraseña

4. El bloque de código devuelve *verdadero*, y la función deja de ejecutarse.
5. En caso contrario, si falla alguna condición, la función devuelve *falso*.
6. Si el formulario fue enviado, se ejecuta el siguiente bloque de código.
7. La contraseña se recoge del superglobal `$_POST`.
8. Se llama a `is_password()` para comprobar la contraseña del usuario. El resultado se almacena en una variable llamada `$valid`.
9. Un operador ternario comprueba si la variable `$valid` es verdadera. En caso afirmativo, `$message` contiene un mensaje de éxito; en caso contrario, contiene un mensaje de error.
10. Se muestra el valor de la variable `$message`.

Ejemplo: comprobar la seguridad de una contraseña

```
<?php
declare(strict_types = 1);
① $password = '';
② $message = '';
③ function is_password(string $password): bool
{
    if (
        mb_strlen($password) >= 8
        and preg_match('/[A-Z]/', $password)
        and preg_match('/[a-z]/', $password)
        and preg_match('/[0-9]/', $password)
    ) {
        ④ return true; // Passed all tests
    }
    ⑤ return false; // Invalid
}
⑥ if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    ⑦ $password = $_POST['password'];
    ⑧ $valid = is_password($password);
    ⑨ $message = $valid ? 'Password is valid' :
        'Password not strong enough';
}
?> ...
⑩ <?= $message ?>
<form action="validate-password.php" method="POST">
    Password: <input type="password" name="password">
    <input type="submit" value="Save">
</form>
```

Ejemplo: comprobar la seguridad de una contraseña





8. Validación de Selecciones y Opciones en Formularios

Cajas de selección y botones de opción

Las casillas de selección (desplegables) y los botones de opción (*radio button*) permiten a los usuarios seleccionar una opción de una lista.

Los navegadores sólo envían el nombre y el valor al servidor si se ha seleccionado una opción.

El valor se valida comprobando si coincide con alguna de las opciones.

Cajas de selección y botones de opción

Cuando un formulario utiliza una caja de selección o botones de opción, puedes crear un array indexado que contenga todas las opciones que el usuario puede elegir, y almacenarlo en una variable.

El siguiente array almacena las puntuaciones de 1-5.

```
$star_ratings = [1, 2, 3, 4, 5,];
```

Dicho array puede ser utilizado para:

- Crear las opciones en casillas de selección o botones de radio
- Comprobar que el usuario ha elegido una de estas opciones

Cajas de selección y botones de opción

Para comprobar que el usuario ha seleccionado una opción válida, se utiliza la función incorporada de PHP `in_array()`.

Si el valor enviado se encuentra en el array de opciones, `in_array()` devuelve *true*. Si no, devuelve *false*.

```
$valid = in_array($stars, $star_ratings);
```



Cajas de selección y botones de opción

Para crear los controles del formulario, se puede recorrer un bucle a través de las opciones y añadir un elemento para cada una de ellas.

Incluso, si se vuelve a mostrar el formulario al usuario, se puede resaltar la opción seleccionada utilizando un operador ternario.

Cajas de selección y botones de opción

En el siguiente ejemplo, la condición de un operador ternario comprueba si el valor de la variable `$stars` coincide con el valor actual del bucle.

- Si es así, se añade el atributo `checked`.
- Si no, en su lugar se escribe una cadena en blanco.

```
<?php foreach ($option as $star_ratings) { ?>
    <?= $option ?>
    <input type="radio" name="stars" value="<?= $option ?>">
        <?= ($stars == $option) ? 'checked' : '' ?>
<?php } ?>
```

Ejemplo: Validando opciones

El siguiente ejemplo valida si la calificación por estrellas seleccionada en un formulario corresponde a una de las opciones válidas (1 a 5) y muestra un mensaje agradeciendo si la selección es válida o solicitando que se elija una opción en caso contrario.

1. Se inicializan las variables `$stars` y `$message`.
2. La variable `$stars_ratings` contiene un array indexado de valores que se utilizará para crear un conjunto de botones de radio.
3. Una sentencia `if` comprueba si el formulario ha sido enviado.

Ejemplo: Validando opciones

4. Si es así, la opción seleccionada se recoge del array superglobal `$_POST`.
5. La función `in_array()` de PHP comprueba si el valor seleccionado por el usuario es una de las opciones permitidas.
6. Se utiliza un operador ternario para crear un mensaje que indique si los datos eran válidos o no.

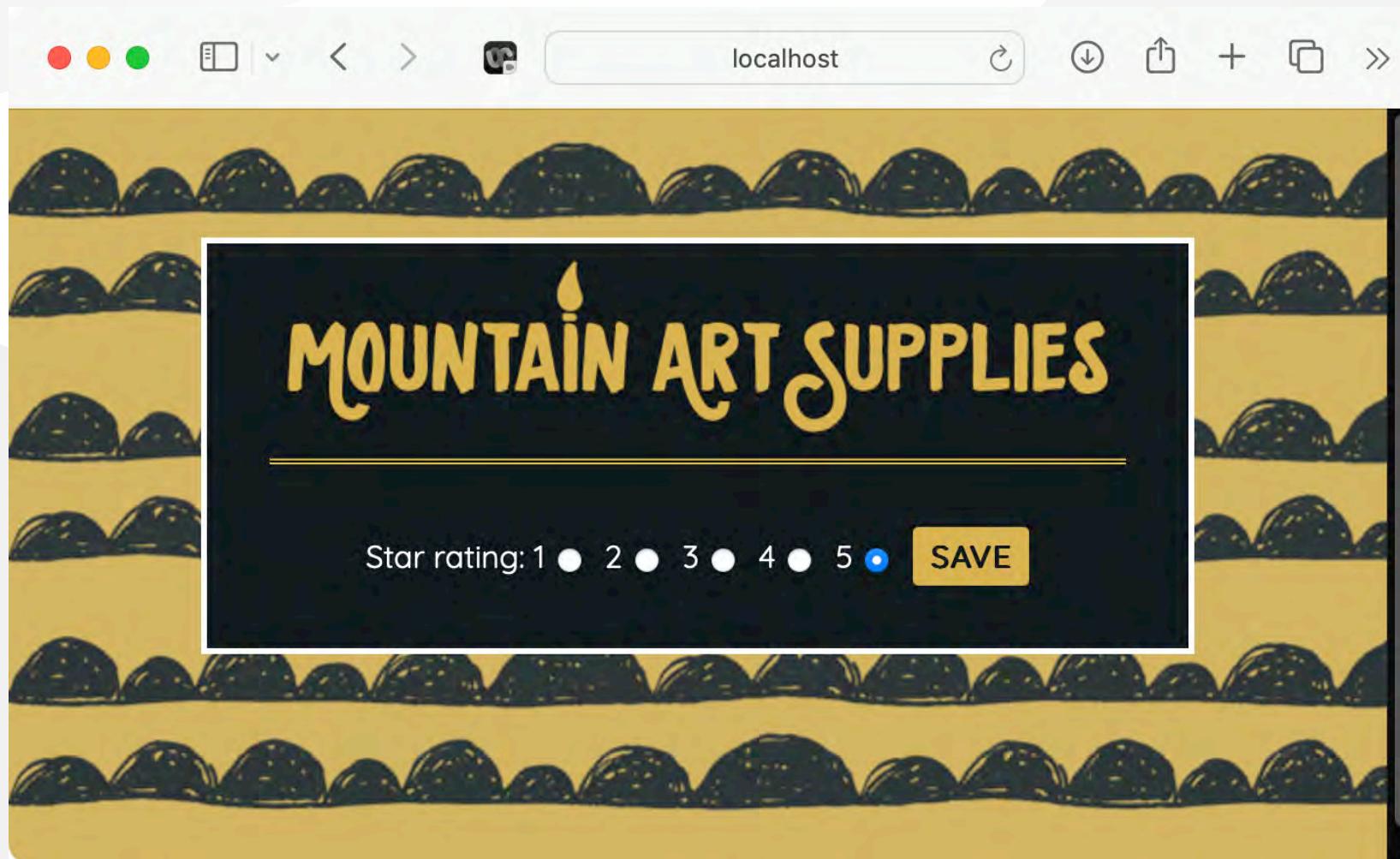
Ejemplo: Validando opciones

7. Se muestra el valor en `$message`.
8. Un bucle `foreach` crea las opciones en el formulario HTML. Trabaja a través de los valores en el array `$star_ratings`. Para cada uno:
 9. Se muestra la opción, seguida de un botón de radio con la opción añadida en el atributo `value`.
 10. Un operador ternario comprueba si se ha seleccionado una opción. En caso afirmativo, se añade el atributo `checked`.

Ejemplo: Validando opciones

```
<?php  
① [ $stars    = '';  
    $message = '';  
② $star_ratings = [1, 2, 3, 4, 5,];  
  
③ if ($_SERVER['REQUEST_METHOD'] == 'POST') {  
④     $stars    = $_POST['stars'] ?? '';  
⑤     $valid    = in_array($stars, $star_ratings);  
⑥     $message = $valid ? 'Thank you' : 'Select an option';  
}  
?> ...  
⑦ <?= $message ?>  
    <form action="validate-options.php" method="POST">  
        Star rating:  
⑧         <?php foreach ($star_ratings as $option) { ?>  
⑨ [             <?= $option ?> <input type="radio" name="stars"  
                value="<?= $option ?>"  
                <?= ($stars == $option) ? 'checked' : '' ?>  
            <?php } ?>  
            <input type="submit" value="Save">  
        </form>
```

Ejemplo: Validando opciones



Ejemplo: Validando opciones

Imagina que estás construyendo un sistema para un instituto en el que los estudiantes deben elegir una asignatura optativa.

- Las asignaturas disponibles son "Matemáticas", "Física", "Historia" y "Arte".
- Los estudiantes solo pueden elegir una de estas asignaturas.

Ejemplo: Validando opciones

Instrucciones:

1. Crear el archivo principal `optativas.php` :

Este archivo contendrá un formulario donde los estudiantes pueden seleccionar una asignatura optativa. El formulario deberá manejar el envío y la validación de la opción seleccionada empleando *radio buttons*.

2. Crear archivos de cabecera y pie de página `header.php` y `footer.php` :

Estos archivos se incluirán en el archivo principal para estructurar la página de manera adecuada.

Ejemplo: Validando opciones

Instrucciones:

3. Tareas a realizar:

- i. Ejecuta `optativas.php` en tu servidor web local.
- ii. Completa el formulario seleccionando una asignatura optativa y envíalo.
- iii. Observa cómo se valida la opción seleccionada y cómo se muestra un mensaje de confirmación o error.
- iv. Modifica el código para que el formulario utilice `cajas de selección` en lugar de un `radio button`, y asegúrate de que la validación siga funcionando correctamente.

Como saber si un *checkbox* está marcado

Una casilla de verificación (o *checkbox*) puede estar marcada o no, pero el nombre y el valor de la casilla sólo se envían al servidor cuando la casilla está marcada.

Determinar si una casilla de verificación fue seleccionada o no, implica dos pasos:

- Primero, utiliza la función `isset()` de PHP para comprobar si hay un valor para la casilla de verificación en el array superglobal.
- Si lo hay, entonces hay que comprobar si el valor proporcionado era el valor que esperaba que se enviara.

Como saber si un *checkbox* está marcado

Ambas comprobaciones pueden realizarse en la condición de un operador ternario. Si ambas comprobaciones resultan en valores de *true*, sabremos que el usuario ha marcado la casilla y podremos asignarle un valor booleano de *true*.

A continuación, si ambas comprobaciones resultan en *true*, `$terms` almacenará el valor *true*, de lo contrario almacenará *false*.

```
$terms = (isset($_POST['terms']) and $_POST['terms'] == true) ? true : false;
```

IF VALUE WAS ADDED TO SUPERGLOBAL ARRAY AND THE VALUE PROVIDED IS CORRECT

Como saber si un *checkbox* está marcado

Si se vuelve a mostrar el formulario al usuario, para que marque una casilla que había seleccionado, basta con comprobar si el valor de esa casilla es verdadero.

Si lo es, entonces el atributo *checked* se añade al control. Si no lo es, en su lugar se escribe una cadena en blanco.

```
<input type="checkbox" name="terms" value="true"  
      <?= $terms ? 'checked' : '' ?>>
```

IF CHECKBOX
WAS SELECTED ADD CHECKED
ATTRIBUTE OTHERWISE ADD
A BLANK STRING

Ejemplo: Validando checkboxes

El siguiente código verifica si el usuario ha marcado la casilla de aceptación de los términos y condiciones en un formulario, mostrando un mensaje de agradecimiento si lo ha hecho, o indicando que debe aceptar los términos si no lo ha hecho.

1. Se inicializan las variables `$terms` y `$message`.
2. Si el formulario ha sido enviado...

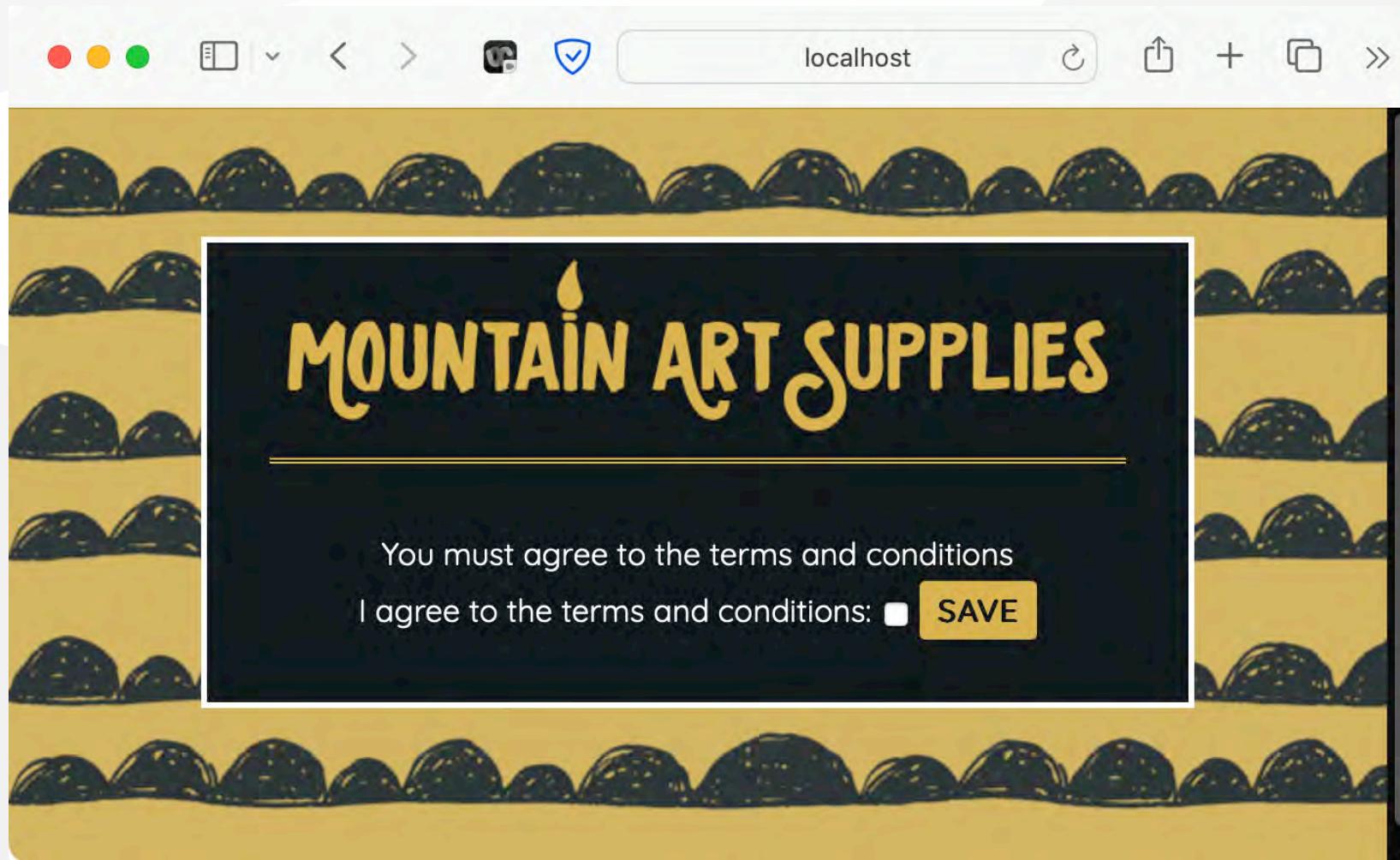
Ejemplo: Validando checkboxes

3. Se utilizan dos expresiones en la condición de un operador ternario para determinar si la casilla de verificación fue marcada. Primero, la función `isset()` de PHP comprueba si el checkbox fue enviado. Si lo fue, la segunda expresión comprueba si su valor es verdadero. Si ambas expresiones se evalúan como *true*, a la variable `$terms` se le asigna el valor *true*; si no, se le asigna el valor *false*.
4. Si `$terms` almacena un valor de *true*, `$message` almacena las palabras *Thank you*; si no, almacena un mensaje indicando al usuario que acepte los términos y condiciones.
5. El mensaje se muestra en la página.
6. Se utiliza un operador ternario para comprobar si la variable `$terms` tiene el valor *true* (lo que indica que se ha comprobado). En caso afirmativo, se añade el atributo `checked` a la casilla de verificación. En caso contrario, se escribe una cadena en

Ejemplo: Validando checkboxes

```
<?php  
① $terms  = '';  
① $message = '';  
  
② if ($_SERVER['REQUEST_METHOD'] == 'POST') {  
③     $terms  = (isset($_POST['terms']) and $_POST['terms'] == true) ? true : false;  
④     $message = $terms ? 'Thank you' : 'You must agree to the terms and conditions';  
    }  
    ?> ...  
⑤ <?= $message ?>  
    <form action="validate-checkbox.php" method="POST">  
        I agree to the terms and conditions: <input type="checkbox" name="terms" value="true"  
        ⑥      <?= $terms ? 'checked' : '' ?>  
        <input type="submit" value="Save">  
    </form>
```

Ejemplo: Validando checkboxes



Ejemplo: Validando checkboxes

El objetivo de esta actividad es que practiquéis cómo validar la selección de *checkboxes* en formularios HTML utilizando PHP.

El contexto de la actividad se centrará en el proceso de inscripción de voluntarios para las Olimpiadas de París 2024.

Ejemplo: Validando checkboxes

Instrucciones:

1. Crear el archivo principal `voluntarios.php` :

Este archivo contendrá un formulario donde los voluntarios pueden seleccionar los eventos en los que desean participar durante las Olimpiadas de París 2024. El formulario deberá manejar el envío y la validación de las opciones seleccionadas.

```
// Lista de eventos disponibles
$eventos = ['Ceremonia de Apertura', 'Atletismo', 'Natación', 'Ciclismo', 'Ceremonia de Clausura'];
```

2. Crear archivos de cabecera y pie de página `header.php` y `footer.php` :

Estos archivos se incluirán en el archivo principal para estructurar la página de manera adecuada.

Ejemplo: Validando checkboxes

3. Tareas a realizar:

- i. Ejecutar `voluntarios.php` en tu servidor web local.
- ii. Completar el formulario seleccionando los eventos en los que desea participar y enviarlo.
- iii. Mostrar cómo se validan las opciones seleccionadas y cómo se muestra un mensaje de confirmación o error.
- iv. Modificar el formulario para agregar más eventos o cambiar la lógica de validación (por ejemplo, exigir la selección de un número mínimo de eventos).

Comprobar si multiples valores son válidos

A menudo, las páginas deben comprobar si varios datos son válidos antes de trabajar con ellos. Por ejemplo, en la mayoría de los formularios se pide a los visitantes que faciliten más de un dato.

Comprobar si multiples valores son válidos

Echa un vistazo al siguiente formulario. En él se pide a los visitantes que faciliten los siguientes datos (utilizando tres tipos de datos):

- **Nombre** (una cadena de entre 2 y 10 caracteres)
- **Edad** (un número entero entre 16 y 65 años)
- **Aceptación de los términos y condiciones** (un valor booleano de verdadero o falso)

Comprobar si multiples valores son válidos

Cuando el usuario envía el formulario, si alguno de los datos no es válido, la página debería:

- No procesar los datos
- Crear mensajes de error indicando al visitante cómo corregir cada uno de los problemas
- Mostrar los valores introducidos por el usuario

Comprobar si multiples valores son válidos

Please correct the following errors:

Name: Ivy

Age: 15

You must be 16-65

I agree to the terms and conditions

You must agree to the terms and conditions

SAVE

The image shows a dark-themed web form with validation errors. At the top, it says "Please correct the following errors:". Below that, there are two input fields: "Name: Ivy" and "Age: 15". To the right of the age field, an error message "You must be 16-65" is displayed. Further down, there is a checkbox labeled "I agree to the terms and conditions" which is not checked, and another error message "You must agree to the terms and conditions" is displayed to its right. At the bottom right of the form is a yellow "SAVE" button.

Comprobar si multiples valores son válidos

Además de mostrar el formulario, esta página puede mostrar mensajes de error y cualquier valor introducido por el usuario.

Para ello, comienza declarando dos arrays:

- Uno con un elemento para contener cada uno de los valores que el usuario proporcionará.
- Uno con un elemento para contener cada uno de los mensajes de error que podrían aparecer en la página.

Comprobar si multiples valores son válidos

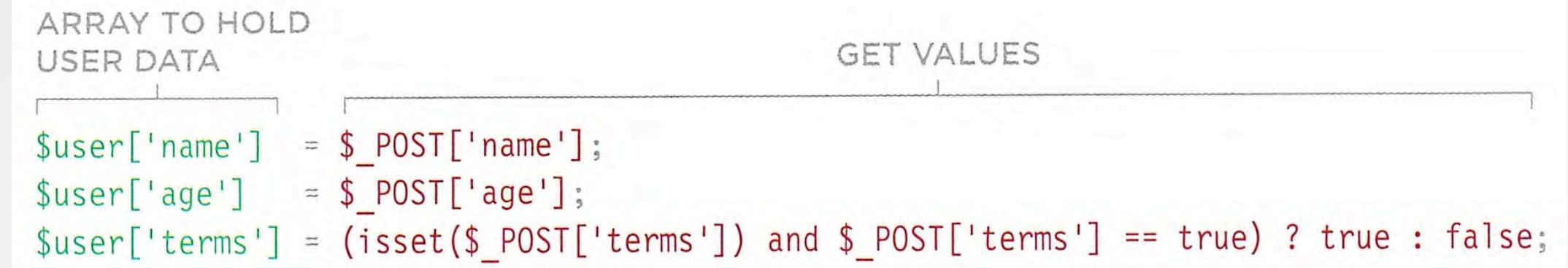
Ambos arrays deben ser inicializados con un nombre para cada elemento y un valor que pueda ser mostrado cuando la página se carga por primera vez (antes de que el formulario haya sido enviado). Si esto no se hiciera, y el intérprete de PHP intentara acceder a un elemento de un array que no tuviera un valor, se produciría un error.

El array que contiene los mensajes de error almacena una cadena en blanco para cada posible error de formulario porque no hay errores cuando la página se carga por primera vez.

Comprobar si multiples valores son válidos

1. Si se ha enviado el formulario, **se recogen los datos que el usuario ha suministrado**.

Estos valores sobrescriben los valores iniciales almacenados en el array que se creó para contener los datos del usuario.



Comprobar si multiples valores son válidos

2. A continuación, **se valida cada dato**. Si no es válido, se almacenará un mensaje de error en el elemento correspondiente del array `$errors`.

Los datos se validan utilizando las funciones de validación que hemos visto en esta unidad, que devuelven true si el valor suministrado por el usuario es válido y false si no lo es.

Esto significa que las funciones de validación pueden llamarse en la condición de un operador ternario (por ejemplo), y si los datos son:

- Válidos: el elemento almacena una cadena en blanco.
- Inválidos: se almacena un mensaje de error indicando al usuario por qué los datos no son válidos.

Comprobar si multiples valores son válidos

ARRAY TO HOLD ERROR MESSAGES	VALIDATE FORM VALUES	BLANK STRING	ERROR MESSAGE
	\$errors['name'] = is_text(\$user['name'], 2, 20)	? ''	: 'Name must be 2-20 characters';
	\$errors['age'] = is_number(\$user['age'], 16, 65)	? ''	: 'You must be 16-65';
	\$errors['terms'] = \$user['terms']	? ''	: 'You must agree to the terms';

Comprobar si multiples valores son válidos

3. Para **comprobar si hubo errores**, se utiliza la función `implode()` incorporada en PHP para unir todos los valores del array `$errors` en una cadena.

El resultado se almacena en una variable llamada `$invalid`. Si `$invalid` contiene una cadena en blanco, los datos eran válidos. Si no, había al menos un error.

```
$invalid = implode($errors);
```

VARIABLE TO HOLD
ALL ERRORS

ARRAY HOLDING
ERRORS

Comprobar si multiples valores son válidos

4. Se utiliza una sentencia if para comprobar si `$invalid` contiene algún texto.
- Si lo hace, se tratará como si fuese un *true* y **se mostrarán los mensajes de error con el formulario**.
 - Si no hubo errores, `$invalid` contendrá una cadena en blanco (que se trata como *false*), y **la página puede procesar los datos que recibió**.

```
if ($invalid) {  
    // Show error messages and do not process the data  
} else {  
    // Data is valid, the page can process the data  
}
```

Ejemplo: Validando formularios

Este ejemplo muestra cómo validar múltiples controles de formulario. El resultado se muestra en la página anterior.

1. `validate.php` está incluido en la página. Contiene las definiciones de tres de las funciones de validación de este capítulo. Colocarlas en un archivo include permite a cualquier página incluir este archivo y luego utilizar sus funciones.
2. La variable `$user` contiene un array con un elemento para cada control del formulario; se les asigna un valor inicial para utilizar en el formulario cuando se carga la página por primera vez.
3. La variable `$errors` almacena un array que tiene un elemento por cada dato que se está validando.

Ejemplo: Validando formularios

4. A `$message` se le asigna una cadena en blanco. Una vez validados los datos, contendrá un mensaje de éxito o error.
5. Una sentencia if comprueba si el formulario ha sido enviado.
6. Si es así, se recogen los tres datos del formulario, y los datos que el usuario proporcionó sobrescriben los valores iniciales que se almacenaron en el array `$user`.
7. El nombre proporcionado por el usuario se valida utilizando la función `is_text()`. Devuelve verdadero si los datos son válidos; falso en caso contrario. Si es válido, el elemento correspondiente en el array `$errors` (ver Paso 3) contiene una cadena en blanco. Si no lo es, contiene un mensaje indicando al usuario cómo solucionarlo.

Ejemplo: Validando formularios

8. La edad del usuario se valida utilizando `is_number()`. Devuelve true si el dato es válido; false en caso contrario. Si es válido, el elemento correspondiente del array `$errors` contiene una cadena en blanco. Si no, contiene un mensaje de error.
9. Si el usuario marcó la casilla de verificación de términos, el elemento correspondiente del array `$errors` contiene una cadena en blanco. Si no, contiene un mensaje de error diciendo que necesitan aceptar los términos y condiciones.

Ejemplo: Validando formularios

10. Todos los valores del array `$errors` se unen en una sola cadena utilizando la función `implode()` de PHP. El resultado se almacena en una variable llamada `$invalid`.
11. La condición de una sentencia if comprueba si el valor en `$invalid` es verdadero. Si contiene algún texto será tratado como verdadero. Una cadena en blanco se trata como falso.
12. Si los datos no son válidos, la variable `$message` almacena un mensaje indicando al usuario que corrija los errores del formulario.
13. En caso contrario, `$message` almacena un mensaje para decir que los datos eran válidos. Si los datos son válidos, la página puede entonces procesarlos. (A menudo, cuando una página ha recibido datos válidos, no será necesario volver a mostrar el formulario).

Ejemplo: Validando formularios

14. Se muestra el valor almacenado en `$message` .
15. Si el usuario ha enviado el formulario, el valor que introdujo para su nombre se escribe en el atributo `value` del control del formulario. (Este texto se escapa utilizando la función `htmlspecialchars()` de PHP). Si el formulario no fue enviado, mostrará la cadena en blanco que fue almacenada en la clave correspondiente del array `$user` cuando fue inicializado en el Paso 2.

Ejemplo: Validando formularios

16. Se muestra el valor del elemento del array `$errors` que corresponde a este control de formulario.
17. Si el usuario proporcionó su edad, se muestra en el atributo value del control de forma. A continuación se muestra el valor correspondiente en el array `$errors`.
18. Si el visitante marcó la casilla de términos y condiciones, se le añade el atributo checked. Le sigue el valor correspondiente en el array `$errors`.

Ejemplo: Validando formularios

```
<?php
declare(strict_types = 1); // Enable strict types
① require 'includes/validate.php'; // Validation functions

② $user = [
    'name' => '',
    'age'  => '',
    'terms' => '',
];
// Initialize $user array

③ $errors = [
    'name' => '',
    'age'  => '',
    'terms' => '',
];
// Initialize errors array

④ $message = '';
// Initialize message
```

Ejemplo: Validando formularios

```
⑤ if ($_SERVER['REQUEST_METHOD'] == 'POST') {                                // If form submitted
    $user['name'] = $_POST['name'];                                         // Get name
    $user['age'] = $_POST['age'];                                            // Get age then check T&Cs
    $user['terms'] = (isset($_POST['terms']) and $_POST['terms'] == true) ? true : false;

⑦ $errors['name'] = is_text($user['name'], 2, 20) ? '' : 'Must be 2-20 characters';
⑧ $errors['age'] = is_number($user['age'], 16, 65) ? '' : 'You must be 16-65';
⑨ $errors['terms'] = $user['terms'] ? '' : 'You must agree to the
    terms and conditions';                                                 // Validate data

⑩ $invalid = implode($errors);                                              // Join error messages
⑪ if ($invalid) {                                                       // If there are errors
    $message = 'Please correct the following errors:';                      // Do not process
    ⑫ } else {                                                               // Otherwise
        $message = 'Your data was valid';                                     // Can process data
    }
}
?> ...
⑭ <?= $message ?>
```

Ejemplo: Validando formularios

```
1 <form action="validate-form.php" method="POST">
2   Name: <input type="text" name="name" value=<?= htmlspecialchars($user['name']) ?>">
3   <span class="error"><?= $errors['name'] ?></span><br>
4   Age:  <input type="text" name="age" value=<?= htmlspecialchars($user['age']) ?>">
5   <span class="error"><?= $errors['age'] ?></span><br>
6   <input type="checkbox" name="terms" value="true" <?= $user['terms'] ? 'checked' : '' ?>
7   I agree to the terms and conditions
8   <span class="error"><?= $errors['terms'] ?></span><br>
9   <input type="submit" value="Save">
10  </form>
```

Ejemplo: Validando formularios

Implementa un formulario de registro de evento donde el usuario debe ingresar su correo electrónico, su edad, y confirmar su interés en recibir boletines (newsletter).



9. Uso de Filtros para Validación y Saneamiento de Datos

Recogida de datos mediante funciones de filtro

PHP también tiene dos funciones incorporadas que recogen los datos enviados desde el navegador y los almacenan en variables. Se llaman **funciones de filtro** (*filter functions*) porque pueden aplicar un filtro a los datos que envió el navegador.

`filter_input()` obtiene un único valor que ha sido enviado al servidor. Requiere dos argumentos. El primer argumento es la **fuente de entrada** (que no se escribe entre comillas). Utiliza:

- **INPUT_GET** para obtener datos enviados a través de HTTP GET.
- **INPUT_POST** para obtener datos enviados a través de HTTP POST.
- **INPUT_SERVER** para obtener los mismos datos que estaban disponibles en el array superglobal `$_SERVER`.

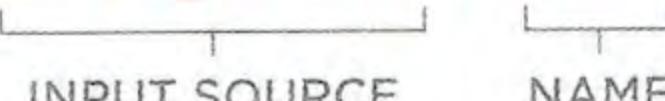
Recogida de datos mediante funciones de filtro

El segundo argumento es el **nombre de un par nombre/valor** enviado al servidor, que debe ir entre comillas. Utilizado de esta forma, `filter_input()` devuelve

- El valor si se envió al servidor
- `null` si los datos no fueron enviados al servidor

Una vez que hayamos aprendido a utilizar esta función, aprenderemos a aplicar un **filtro** como tercer parámetro.

```
$data = filter_input(INPUT_SOURCE, 'name');
```



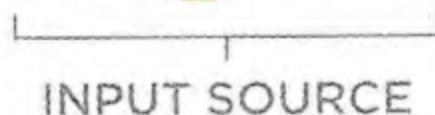
INPUT SOURCE NAME

Recogida de datos mediante funciones de filtro

`filter_input_array()` recoge todos los valores que se enviaron al servidor a través de HTTP GET o POST y almacena cada uno de ellos como elemento de un array.

Como obtiene todos los valores, sólo necesita un argumento: la fuente de entrada. Los valores para la fuente de entrada son los mismos que para `filter_input()`.

```
$data = filter_input_array(INPUT_SOURCE);
```



INPUT SOURCE

Recogida de datos mediante funciones de filtro

Cuando se reciben los datos, se almacenan como un tipo de datos de cadena (*string*). Algunos de los filtros que se pueden aplicar utilizando estas funciones de filtro convertirán el tipo de datos.

Los siguientes ejemplos utilizan la función `var_dump()` de PHP para mostrar los valores recogidos utilizando estas funciones, ya que es importante ver los tipos de datos de cada valor.

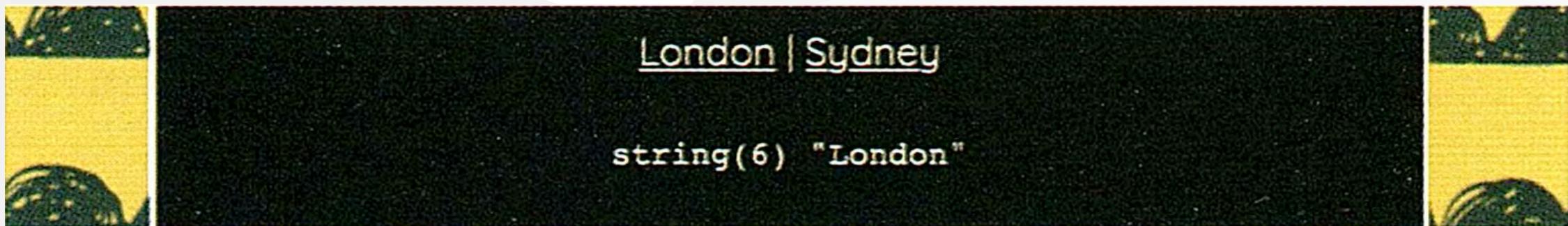
Ejemplo: Funciones de filtro

Cuando estos dos ejemplos siguientes se cargan por primera vez, el valor mostrado será *NULL* ya que la cadena de consulta está vacía.

1. La función `filter_input()` recoge un único valor enviado a través de HTTP GET en la cadena de consulta. El nombre del par nombre/valor es `city`. El valor recogido se almacena en una variable llamada `$location`.
2. Dos enlaces utilizan cadenas de consulta para enviar un nombre llamado `city`; sus valores son diferentes nombres de ciudades.
3. Se utiliza `var_dump()` para mostrar el valor almacenado en `$location` y su tipo de datos (una cadena).

Ejemplo: Funciones de filtro

```
① <?php $location = filter_input(INPUT_GET, 'city'); ?> ...
② [ <a href="filter_input.php?city=London">London</a> |  
  <a href="filter_input.php?city=Sydney">Sydney</a>
③ <pre><?php var_dump($location); ?></pre>
```



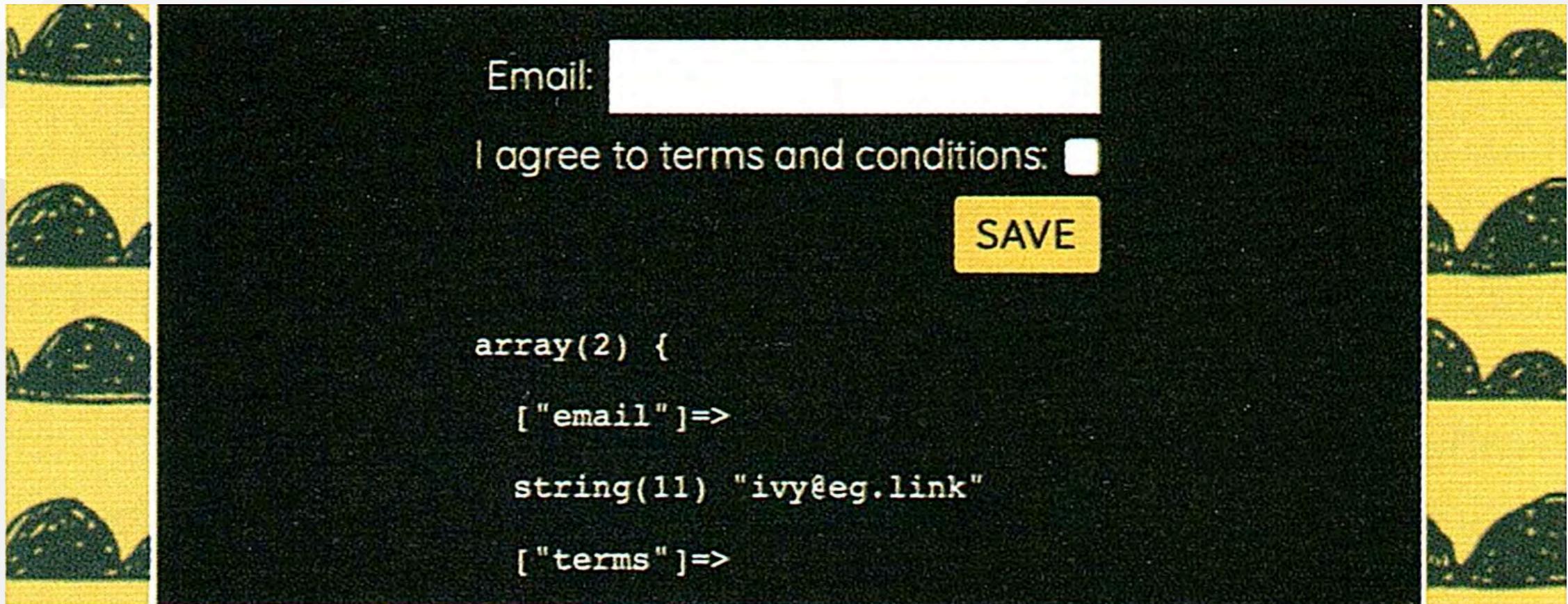
Ejemplo: Funciones de filtro

4. La función `filter_input_array()` se utiliza para obtener todos los valores del formulario cuando se envía mediante HTTP POST. El array que crea la función se almacena en una variable llamada `$form`.
5. El formulario envía una entrada de texto y una casilla de verificación a través de HTTP POST.
6. `var_dump()` muestra los nombres y valores almacenados en `$form` junto con el tipo de datos de cada valor.

Ejemplo: Funciones de filtro

```
④ <?php $form = filter_input_array(INPUT_POST); ?> ...
<form action="filter_input_array.php" method="POST">
    Email: <input type="text" name="email" value=""><br>
    I agree to terms and conditions:
    <input type="checkbox" name="terms" value="true"><br>
    <input type="submit" value="Save">
</form>
⑥ <pre><?php var_dump($form); ?></pre>
```

Ejemplo: Funciones de filtro



Ejemplo: Funciones de filtro

Adapta el ejercicio anterior implementando funciones de filtro:

"Implementa un formulario de registro de evento donde el usuario debe ingresar su correo electrónico, su edad, y confirmar su interés en recibir boletines (newsletter)."

Filtros de validación

Cuando las funciones de filtro obtienen datos enviados desde un navegador, se almacenan como una cadena.

A continuación puedes ver tres filtros de validación que comprueban si un valor es un booleano, un entero o un flotante.

Cada filtro internamente tiene un ID de filtro que se utiliza para identificarlo.

Filtros de validación

Si una página espera recibir un valor que es un booleano, un entero o un flotante, las funciones de filtro pueden utilizar los tres filtros siguientes para comprobar si el valor proporcionado es el tipo de datos correcto.

Cuando estos filtros comprueban si un valor es un booleano, un entero o un flotante, las funciones de filtro convierten el valor del tipo de datos cadena al tipo de datos especificado en el filtro (veremos cómo utilizar estos filtros a continuación).

FILTER ID	DESCRIPTION
FILTER_VALIDATE_BOOLEAN	Checks if a value is true. 1, on, and yes all count as true. It is not case-sensitive. If it is true, the function returns a boolean value of true. If not, it returns false.
FILTER_VALIDATE_INT	Checks if a number is an integer (0 is not counted as a valid integer). If valid, it returns the number as an int data type. If not, it returns false.
FILTER_VALIDATE_FLOAT	Checks if a number is a floating point number (decimal). Integers pass the filter (0 does not as it is not counted as a valid integer). If valid, it returns the value as a float data type. If not, it returns false.

Filtros de validación

Cada filtro también tiene dos tipos de ajustes que se pueden utilizar para controlar cómo se comporta el filtro:

- Los **indicadores** son ajustes que se pueden activar o desactivar.
- Las **opciones** son ajustes en los que debe establecer un valor.

Por ejemplo, los filtros integer y float tienen opciones que te permiten especificar el número mínimo y máximo que el usuario puede proporcionar. Así, si se pide a un visitante que indique su edad y ésta debe estar comprendida entre 16 y 65 años, el filtro puede comprobar si el número facilitado está dentro de este intervalo. Si el número no se proporciona, o si el número es demasiado bajo o demasiado alto, sería inválido.

Filtros de validación

Todos los filtros de validación tienen también una opción que permite especificar un **valor por defecto** que debe utilizarse si los datos recibidos no son válidos.

Las **banderas o indicadores (flags)** son opciones que sólo pueden activarse. Por ejemplo, el filtro de enteros tiene una bandera que puede activar para permitir a los visitantes proporcionar números utilizando notación hexadecimal además de los dígitos estándar 0-9. (La notación hexadecimal utiliza los dígitos 0-9 y las letras A-F para representar números 10-15; puede que hayas visto que se utiliza para especificar colores en HTML y CSS).

Más adelante se muestra una lista completa de los filtros de validación, junto con sus indicadores y opciones.

Filtros de validación

A continuación, puedes ver los filtros de validación que se utilizan para recopilar texto. También se pueden utilizar expresiones regulares para escribir filtros personalizados.

Los datos suelen seguir reglas relativas a:

- El número de caracteres que pueden contener
- Los caracteres que pueden utilizar
- El orden en que pueden aparecer esos caracteres

Por ejemplo, hay reglas que controlan cómo se utilizan los caracteres en direcciones de correo electrónico, URLs, nombres de dominio y direcciones IP. Los cuatro filtros siguientes comprueban si un valor sigue estas reglas.

Filtros de validación

FILTER ID	DESCRIPTION
<code>FILTER_VALIDATE_EMAIL</code>	Checks if the structure of a string matches that of an email address.
<code>FILTER_VALIDATE_URL</code>	Checks if the structure of a string matches that of a URL.
<code>FILTER_VALIDATE_DOMAIN</code>	Checks if the structure of a string matches that of a valid domain name.
<code>FILTER_VALIDATE_IP</code>	Checks if the structure of a string matches that of a valid IP address.

Filtros de validación

Las expresiones regulares pueden utilizarse para escribir otros filtros que comprueben si un valor contiene un patrón de caracteres.

La expresión regular se especifica como opción para el filtro `FILTER_VALIDATE_REGEXP`.

FILTER ID	DESCRIPTION
<code>FILTER_VALIDATE_REGEXP</code>	Checks if a string contains a pattern of characters described using a regular expression (see p212-15).

Utilizando filtros para validar datos individuales

Cuando se utilizan las funciones de filtro para comprobar los datos, se les debe indicar el ID del filtro que deben utilizar, así como cualquier indicador u opción que deba seguir el filtro.

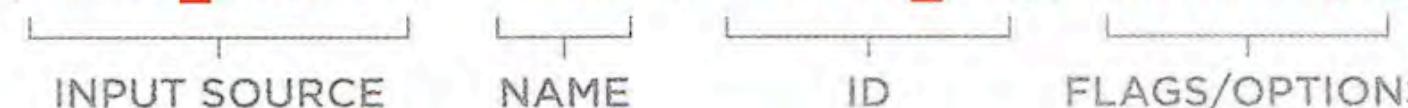
Utilizando filtros para validar datos individuales

Cuando se utiliza `filter_input()` para recoger un dato individual, el **tercer parámetro** es el ID del filtro que se va a utilizar, y el **cuarto parámetro (opcional)** contiene los ajustes que puede utilizar el filtro.

La función `filter_input()` devuelve:

- El valor que recibió si pasa el filtro
- `false` si no pasa el filtro
- `null` si no se envió el nombre al servidor

```
$data = filter_input(INPUT_SOURCE, 'name', FILTER_ID[, $settings]);
```



The diagram illustrates the parameters of the `filter_input` function. It shows four parameters: `INPUT_SOURCE`, `NAME`, `ID`, and `FLAGS/OPTIONS`. The first three parameters are grouped together under a bracket, indicating they are required. The fourth parameter is also grouped under a bracket, indicating it is optional.

Utilizando filtros para validar datos individuales

Cuando un filtro utiliza banderas y opciones, se almacenan en un array asociativo, con dos claves:

- `flags` contiene los parámetros que pueden activarse
 - `options` contiene las opciones que requieren un valor

En el siguiente ejemplo, un array de flags y options se almacena en una variable llamada `$settings`.



Utilizando filtros para validar datos individuales

El valor de la clave `flags` es el nombre de la opción que debe activarse (no se escribe entre comillas). Para utilizar varias banderas, se debe separar el nombre de cada una de ellas con el carácter `|`.

El valor para la clave `options` es otro array asociativo; la clave de cada elemento es el nombre de la opción que se está configurando y el valor es el valor a utilizar.

NOTA: Cuando `filter_input()` recoge datos que no son válidos, devuelve `false`, lo que significa que el valor suministrado por el usuario no puede mostrarse en un formulario.

Ejemplo: recolectando datos usando filtros

1. La variable `$settings` contiene un array de banderas y opciones utilizadas para validar un número. La bandera permite dar el número en notación hexadecimal. Las opciones establecen que el número mínimo permitido es 0 y el número máximo permitido es 255.
2. La función `filter_input()` obtiene el valor enviado a través de HTTP POST desde un control de formulario cuyo nombre es `number`. El tercer parámetro es el ID del filtro. El cuarto parámetro es el nombre de la variable que contiene el array de opciones y flags a utilizar con el filtro.

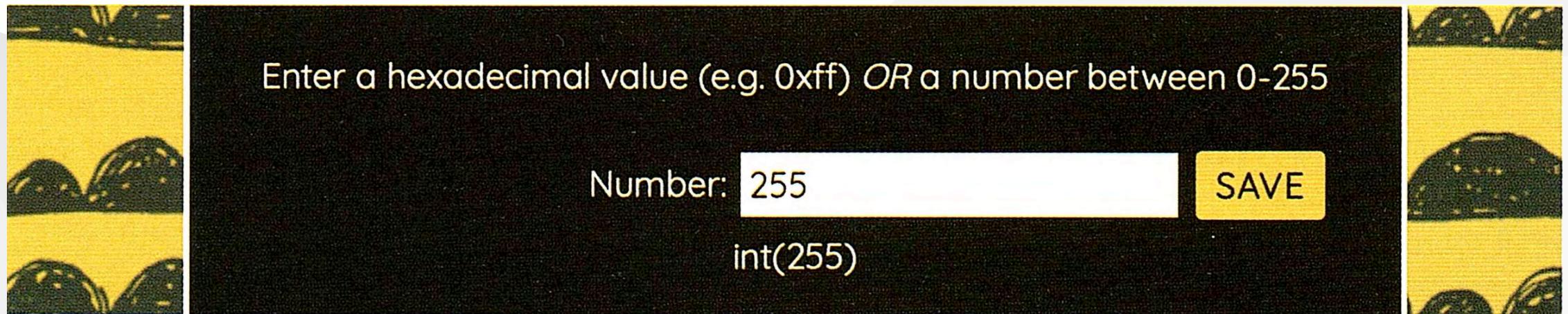
Ejemplo: recolectando datos usando filtros

3. El valor almacenado en `$number` se muestra en el control del formulario. Si el valor en `$number` es null (porque el formulario no fue enviado) o false (porque los datos no eran válidos), no lo verás mostrado en el control del formulario. Esto se debe a que PHP no muestra nada para valores falsos o nulos.
4. `var_dump()` se utiliza para mostrar el valor almacenado en `$number` (porque false o null no se muestran en un navegador). También muestra el tipo de datos porque todos los números válidos son convertidos de cadenas a enteros.

Ejemplo: recolectando datos usando filtros

```
<?php  
① [ $settings['flags'] = FILTER_FLAG_ALLOW_HEX; // Allow hex flag  
    $settings['options']['min_range'] = 0; // Min number option  
    $settings['options']['max_range'] = 255; // Max number option  
  
② $number = filter_input(INPUT_POST, 'number', FILTER_VALIDATE_INT, $settings);  
?  
    ...  
    <form action="validate-input.php" method="POST">  
③    Number: <input type="text" name="number" value=<?= htmlspecialchars($number) ?>>  
        <input type="submit" value="Save">  
    </form>  
④ <?php var_dump($number); ?>
```

Ejemplo: recolectando datos usando filtros



Ejemplo: recolectando datos usando filtros

Implementa un formulario para que el usuario ingrese una dirección IP o un rango de direcciones IP válidas.

- Sólo se permitirán direcciones IPv4 y sin rangos reservados (evitar direcciones reservadas como las de red privada o loopback).
- El valor por defecto si una IP no es válida será "0.0.0.0".

Filtros para validar multiples inputs

Para recoger y validar un conjunto de valores al mismo tiempo, puede utilizar `filter_input_array()`, y especificar un filtro a utilizar para cada dato que se recoja.

Filtros para validar multiples inputs

Cuando una página espera recibir múltiples valores, crea **un array asociativo con un elemento por cada valor que la página espera recibir**. Las claves de cada elemento del array son los **nombres de los controles del formulario** o los **nombres de la cadena de consulta**.

El valor de cada elemento es:

- El **nombre del filtro** que se utilizará cuando se recopilen los datos (**si no tiene flags u opciones**).
- **Un array** que contiene el nombre del filtro y las banderas u opciones que debe utilizar

Filtros para validar multiples inputs

```
$filters['name1'] = FILTER_ID;  
$filters['name2']['filter'] = FILTER_ID;  
$filters['name2']['options']['option1'] = value1;  
$filters['name2']['options']['option2'] = value2;
```

Filtros para validar multiples inputs

A continuación, se llama a la función `filter_input_array()` con dos parámetros:

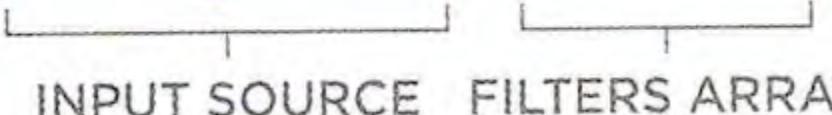
- La **fuente** de entrada (`INPUT_GET` o `INPUT_POST`).
- **Array de filtros** que deben utilizarse con los valores que la página espera recibir para cada entrada.

Esta función (`filter_input_array()`) devuelve un nuevo array asociativo, donde la clave de cada elemento es el nombre de la entrada, y el valor es:

- El **valor** proporcionado *si es válido*.
- **false** si el valor se proporcionó, pero *no es válido*.
- **null** si *no se proporcionó el nombre*.

Filtros para validar multiples inputs

```
$data = filter_input_array(INPUT_SOURCE, $filters);
```



Filtros para validar multiples inputs

Si la página recibe datos adicionales que no se especificaron en el array de filtros, **esos datos no se añaden al array que devuelve `filter_input_array()`**.

Si falta algún dato, se le asigna el valor *null*. **Para evitar que los valores que faltan se añadan al array resultante**, se debe especificar *false* como tercer argumento.

Ejemplo: Filtros para validar multiples inputs

1. El array `$form` se inicializa con valores para las entradas `email` y `age`.
2. Si el formulario ha sido enviado, `$filters` almacena un array. La clave de cada elemento es el nombre de un control de formulario. Los valores son los **filtros/opciones a utilizar**:
 - `email` debe seguir el formato de una dirección de correo electrónico.
 - la edad (`age`) debe ser un número entero de 16 años o más.

Ejemplo: Filtros para validar multiples inputs

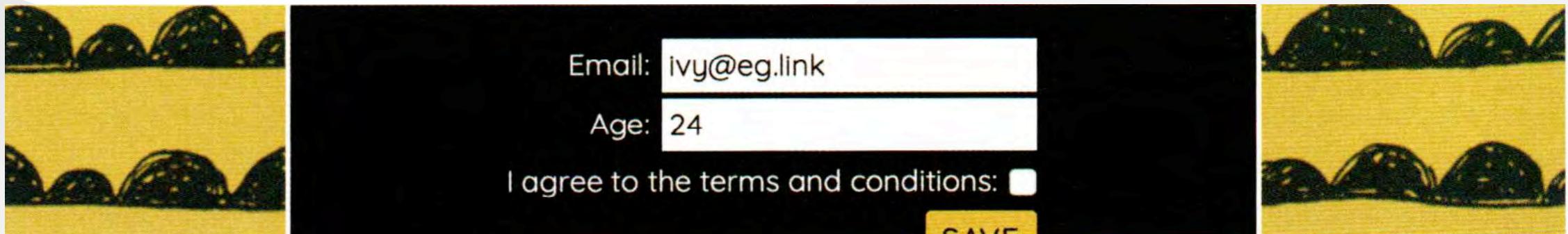
3. `filter_input_array()` recoge y valida los datos y sobrescribe los valores almacenados en `$form`.
4. La función `var_dump()` muestra los datos.

NOTA: Cuando se envía el formulario, aunque la casilla de verificación de términos esté marcada, no se añade al array `$form` ya que no se nombró en el array `$filters`. Además, los datos no válidos no se muestran en el control del formulario.

Ejemplo: Filtros para validar multiples inputs

```
<?php
① $form['email'] = '';
    $form['age'] = '';
    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        ② $filters['email']           = FILTER_VALIDATE_EMAIL; // Email filter
        $filters['age']['filter']   = FILTER_VALIDATE_INT; // Integer filter
        $filters['age']['options']['min_range'] = 16; // Min value 16
        ③ $form = filter_input_array(INPUT_POST, $filters); // Validate data
    }
    ?> ...
<form action="validate-multiple-inputs.php" method="POST">
    Email: <input type="text" name="email" value="<?= htmlspecialchars($form['email']) ?>">
    Age: <input type="text" name="age" value="<?= htmlspecialchars($form['age']) ?>"><br>
    I agree to the terms and conditions: <input type="checkbox" name="terms" value="1"><br>
    <input type="submit" value="Save">
</form>
④ <pre><?php var_dump($form); ?></pre>
```

Ejemplo: Filtros para validar multiples inputs



The image shows a screenshot of a web application interface. It features a black header bar at the top. Below the header, there are two input fields: one for email with the value "ivy@eg.link" and one for age with the value "24". Below these fields is a statement "I agree to the terms and conditions:" followed by a small square checkbox. At the bottom right of the form area is a yellow button labeled "SAVE". The background of the page is white, and there are decorative green and yellow patterns on the left and right sides.

Ejemplo: Filtros para validar multiples inputs

1. Crea un formulario en HTML con los siguientes campos:

- Email.
- Edad (entre 18 y 65 años).
- URL de un sitio web.
- Aceptar términos y condiciones (checkbox).

2. Valida los inputs en PHP usando `filter_input_array()`.

- Ingresa diferentes valores en el formulario para probar la validación. Muestra qué sucede cuando ingresas valores incorrectos (por ejemplo, un email inválido o una edad fuera del rango permitido).



10. Filtros sobre variables: Flags y Opciones Avanzadas

Funciones de filtro para trabajar con variables

PHP tiene dos funciones de filtro incorporadas para filtrar valores almacenados en variables.

- `filter_var()` aplica un filtro a un único valor almacenado en una variable.
- `filter_var_array()` aplica filtros a un conjunto de valores almacenados en un array.

Funciones de filtro para trabajar con variables

La función `filter_var()` requiere:

- Nombre de la variable cuyo valor se comprobará
- ID del filtro

The diagram shows the `filter_var` function signature: `filter_var($variable, FILTER_ID[, $settings])`. Brackets below the parameters indicate their purpose: the first bracket covers `$variable` and `FILTER_ID`, labeled "VARIABLE HOLDING DATA" and "FILTER" respectively; the second bracket covers `$settings`, labeled "FLAGS/OPTIONS".

```
filter_var($variable, FILTER_ID[, $settings]);
```

VARIABLE HOLDING DATA FILTER FLAGS/OPTIONS

Los valores de las opciones o banderas se establecen de la misma forma que en `filter_input()`. Los valores que devuelve también son los mismos: si es válido, devuelve el valor; si no es válido, devuelve `false`; si falta, devuelve `null`.

Funciones de filtro para trabajar con variables

La función `filter_var_array()` también tiene dos parámetros:

- El nombre de la variable que contiene un array cuyos datos se comprobarán.
- El array de filtros y sus opciones/banderas

The diagram shows the function signature `filter_var_array($array, $filters)`. Below it, two horizontal brackets with arrows point to the parameters. The first bracket points to `$array` with the label "VARIABLE HOLDING ARRAY" underneath. The second bracket points to `$filters` with the label "FILTERS TO USE" underneath.

Los valores para las opciones o banderas se establecen de la misma forma que para `filter_input_array()` y los valores que se devuelven también son los mismos.

Si sólo se nombra un filtro, se aplica el mismo filtro a todos los valores del array.

Funciones de filtro para trabajar con variables

Cuando se utiliza `filter_input()` o `filter_input_array()` para validar datos, devuelven *false* para cualquier dato inválido (reemplazando el valor que el usuario envió).

Esto significa que si el formulario contiene datos inválidos, el usuario no podrá ver ningún valor inválido que haya introducido en el formulario.

Para mostrar valores incorrectos, hay que recoger los datos y almacenarlos en una variable o array. Luego, cuando se valida utilizando `filter_var()` o `filter_var_array()`, el resultado se puede almacenar en una nueva variable.

Por tanto, si los datos son:

- **válidos**: la página utiliza los datos en la nueva variable.
- **no válidos**: el formulario muestra los datos tal y como se recogieron originalmente antes de ser validados.

Ejemplo: validando datos en variables

Este ejemplo es similar al de la sección anterior:

1. Los arrays `$form` y `$data` se inicializan con valores para mostrar si el formulario no se ha enviado.
2. Si el formulario ha sido enviado, el array `$filters` almacenará los filtros y opciones para validar los datos.
3. `filter_input_array()` recoge los datos del formulario, sobrescribiendo los valores almacenados en `$form` en el paso 1.
4. `filter_var_array()` valida los datos del formulario (utilizando los filtros especificados en la matriz `$filters`). Almacena el array de resultados en una variable llamada `$data`.

Ejemplo: validando datos en variables

5. Las entradas de texto muestran los valores que el usuario suministró (que se almacenaron en el array `$form` antes de validar los datos).
6. La función `var_dump()` muestra los datos que han sido validados y almacenados en el array `$data` (o null si el formulario no fue enviado).

Ejemplo: validando datos en variables

```
<?php
    $form['email'] = '';
    $form['age'] = '';
    $form['terms'] = 0;
    $data = [];

    ① if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $filters['email'] = FILTER_VALIDATE_EMAIL; // Email filter
        $filters['age']['filter'] = FILTER_VALIDATE_INT; // Integer filter
        $filters['age']['options']['min_range'] = 16; // Min age
        $filters['terms'] = FILTER_VALIDATE_BOOLEAN; // Boolean filter
    }

    ② $form = filter_input_array(INPUT_POST); // Get all values
    ③ $data = filter_var_array($form, $filters); // Apply filters
}

?> ...

<form action="validate-variables.php" method="POST">
    ④ Email: <input type="text" name="email" value="<?= htmlspecialchars($form['email']) ?>">
    Age: <input type="text" name="age" value="<?= htmlspecialchars($form['age']) ?>"><br>
    I agree to the terms and conditions: <input type="checkbox" name="terms" value="1"><br>
    <input type="submit" value="Save">
</form>
⑤ <pre><?php var_dump($data); ?></pre>
```

Ejemplo: validando datos en variables

Adapta el ejercicio anterior (*Filtros para validar multiples inputs*) para implementar validación de datos en variables de forma similar a como se ha mostrado en el ejemplo.

Filtros de validación, flags y opciones

Las tablas siguientes muestran los filtros, indicadores y opciones más utilizadas para trabajar con tipos de datos booleanos, numéricos y cadenas de caracteres.

Filtros de validación, flags y opciones

FILTER_VALIDATE_BOOLEAN

Comprueba si un valor es verdadero (1, on o yes se tratan como verdadero). Devuelve un valor booleano de true si el valor es verdadero, false si no lo es, null si el nombre no estaba presente. No distingue entre mayúsculas y minúsculas.

FLAG	DESCRIPTION
FILTER_NULL_ON_FAILURE	Returns null (not false) if invalid

Filtros de validación, flags y opciones

FILTER_VALIDATE_INT

Comprueba si un número es un entero (0 no se considera un entero válido). Si es válido, devuelve el número como un tipo de datos int.

FLAG	DESCRIPTION
<code>FILTER_FLAG_ALLOW_HEX</code>	Allow hexadecimal numbers
<code>FILTER_FLAG_ALLOW_OCTAL</code>	Allow octal numbers
OPTION	DESCRIPTION
<code>min_range</code>	Minimum allowed number
<code>max_range</code>	Maximum allowed number

Filtros de validación, flags y opciones

FILTER_VALIDATE_FLOAT

Comprueba si un número es un número de coma flotante (decimal). Los números enteros son válidos (pero 0 no se considera un número entero válido). Si es válido, devuelve el valor como un tipo de datos float.

FLAG	DESCRIPTION
FILTER_FLAG_ALLOW_THOUSAND	Allows float to have thousand separator Returns null (not false) if invalid

Filtros de validación, flags y opciones

FILTER_VALIDATE_REGEXP

Comprueba si una cadena contiene un patrón de caracteres descrito en una expresión regular.

OPTION	DESCRIPTION
regexp	The regular expression to use

Filtros de validación, flags y opciones

FILTER_VALIDATE_EMAIL

Comprueba si la estructura de una cadena coincide con la de una dirección de correo electrónico.

FLAG	DESCRIPTION
FILTER_FLAG_EMAIL_UNICODE	Allows unicode characters in name part of address (part before @ symbol)

Filtros de validación, flags y opciones

FILTER_VALIDATE_URL

Comprueba si la estructura de una cadena coincide con la de una URL válida.

FLAG	DESCRIPTION
FILTER_FLAG_SCHEME_REQUIRED	Must contain a scheme e.g., http:// or ftp://
FILTER_FLAG_HOST_REQUIRED	Must contain a hostname
FILTER_FLAG_PATH_REQUIRED	Must contain a path to file or directory
FILTER_FLAG_QUERY_REQUIRED	Must contain a query string

Filtros de validación, flags y opciones

FILTER_VALIDATE_DOMAIN

Comprueba si la estructura de una cadena coincide con la de un nombre de dominio.

FLAG	DESCRIPTION
FILTER_FLAG_HOSTNAME	Validates a hostname

Filtros de validación, flags y opciones

FILTER_VALIDATE_IP

Comprueba si la estructura de una cadena coincide con la de una dirección IP válida.

FLAG	DESCRIPTION
FILTER_FLAG_IPV4	Checks if it is a valid IPV4 IP address
FILTER_FLAG_IPV6	Checks if it is a valid IPV6 IP address
FILTER_FLAG_NO_RES_RANGE	Doesn't allow IPs from reserved range (addresses used on local networks and not sent across the Internet)
FILTER_FLAG_NO_PRIV_RANGE	Doesn't allow IPs from private range (a subset of reserved IP addresses)