

Desarrollo de aplicaciones web con PHP.

Caso práctico

Va siendo hora de comenzar a dar pasos firmes en el desarrollo del nuevo proyecto, y en **BK Programación** ultiman los preparativos y se toman las últimas decisiones para establecer el método que seguirán en su desarrollo.

Mientras, **Carlos** revisa los conocimientos adquiridos y decide hacer una lista con todo lo aprendido hasta el momento y otra con todo lo que aún no sabe hacer. El saldo final es bastante positivo: ya sabe utilizar variables, hacer llamadas a funciones, estructurar el código, utilizar formularios web, trabajar con bases de datos... ¡Incluso ha diseñado una pequeña página para gestionar su colección de comics!



Entre lo que no sabe hacer todavía hay algo que le llama la atención: no sabe cómo mantener los datos entre las distintas páginas de una aplicación. Es decir, la única forma que conoce es utilizando un formulario web..., o bases de datos. Pero tiene que haber otras maneras más adecuadas. Algun modo sencillo de mantener información del usuario dentro de la aplicación web.

De nuevo tendrá que pedir consejo a **Juan**.



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Autentificación de usuarios y control de acceso.

Caso práctico

Tal y como **Esteban** les ha explicado, el objetivo principal del proyecto no es crear una página web pública, con información sobre la empresa. Necesitan una aplicación web con un objetivo más específico: permitir a clientes y a empleados conocer información sobre los productos de la empresa.



[Cangjie6 \(CC BY-SA\)](#)

Juan sabe que con estas condiciones, uno de los puntos fundamentales con los que deberá tratar en el nuevo proyecto es el control de acceso a la aplicación web. Todos los usuarios que accedan habrán de identificarse para poder acceder a las páginas del sitio web. Además, en función de si el usuario es un cliente o un empleado, habrá que darle acceso a una o a otra información.

Juan nunca ha programado sitios web con autentificación de los usuarios. Además, se encuentra terminando otro proyecto, y no dispone de demasiado tiempo. Por este motivo, le pide a **Carlos** que se documente sobre el tema para poder decidir el camino a tomar.

Muchas veces es importante verificar la identidad de los dos extremos de una comunicación. En el caso de una comunicación web, existen métodos para identificar tanto al servidor en el que se aloja el sitio web, como al usuario del navegador que se encuentra en el otro extremo.

Los sitios web que necesitan emplear identificación del servidor, como las tiendas o los bancos, utilizan el protocolo HTTPS. Este protocolo requiere de un certificado válido, firmado por una autoridad confiable, que es verificado por el navegador cuando se accede al sitio web. Además, HTTPS utiliza métodos de cifrado para crear un canal seguro entre el navegador y el servidor, de tal forma que no se pueda interceptar la información que se transmite por el mismo.

Para identificar a los usuarios que visitan un sitio web, se pueden utilizar distintos métodos como el DNI digital o certificados digitales de usuario, pero el más extendido es solicitar al usuario cierta información que solo él conoce: la combinación de un nombre de usuario y una contraseña.

En la unidad anterior aprendiste a utilizar aplicaciones web para gestionar información almacenada en bases de datos. En la mayoría de los casos es importante implantar en este tipo de aplicaciones web, las que acceden a bases de datos, algún mecanismo de control de acceso que obligue al usuario a identificarse. Una vez identificado, se puede limitar el uso que puede hacer de la información.

Así, puede haber sitios web en los que los usuarios autenticados pueden utilizar sólo una parte de la información (como los bancos, que permiten a sus clientes acceder únicamente a la información relativa a sus cuentas). Otros sitios web necesitan separar en grupos a los usuarios autenticados, de tal forma que la información a la que accede un usuario depende del grupo en que éste se encuentre. Por ejemplo, una aplicación de gestión de una empresa puede tener un grupo de usuarios a los que permite visualizar la información, y otro grupo de usuarios que, además de visualizar la información, también la pueden modificar.

Debes distinguir la autentificación de los usuarios y el control de acceso, de la utilización de mecanismos para asegurar las comunicaciones entre el usuario del navegador y el servidor web. Aunque ambos aspectos suelen ir unidos, son independientes.

En los ejemplos de esta unidad, la información de autentificación (nombre y contraseña de los usuarios) se envía en texto plano desde el navegador hasta el servidor web. **Esta práctica es altamente insegura y nunca debe usarse sin un protocolo como HTTPS que permita cifrar las comunicaciones con el servidor web.** Sin embargo, la configuración de servidores web que permitan usar el protocolo HTTPS para cifrar la información que reciben y transmiten no forma parte de los contenidos de este módulo. Por este motivo, durante esta unidad utilizaremos únicamente el protocolo no seguro HTTP.

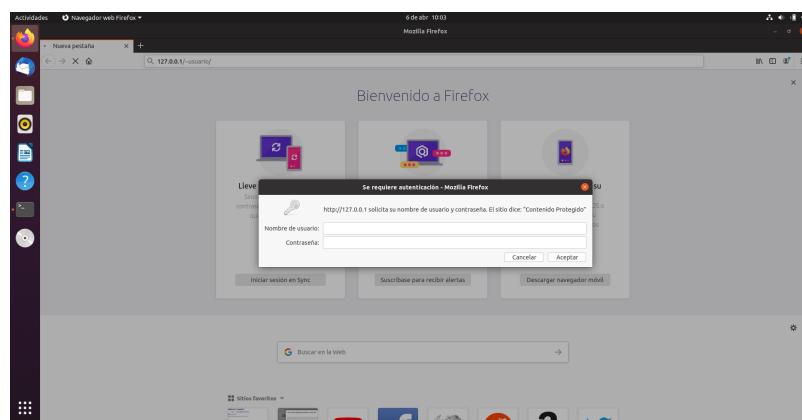
1.1.- Mecanismos de autenticación (I).

El protocolo HTTP ofrece un método sencillo para autenticar a los usuarios. El proceso es el siguiente:

- ✓ El servidor web debe proveer algún método para definir los usuarios que se utilizarán y cómo se pueden autenticar. Además, se tendrán que definir los recursos a los que se restringe el acceso y qué regla (ACL) se aplica a cada uno.
- ✓ Cuando un usuario no autenticado intenta acceder a un recurso restringido, el servidor web responde con un error de "Acceso no autorizado" (código 401).
- ✓ El navegador recibe el error y abre una ventana para solicitar al usuario que se autentifique mediante su nombre y contraseña.

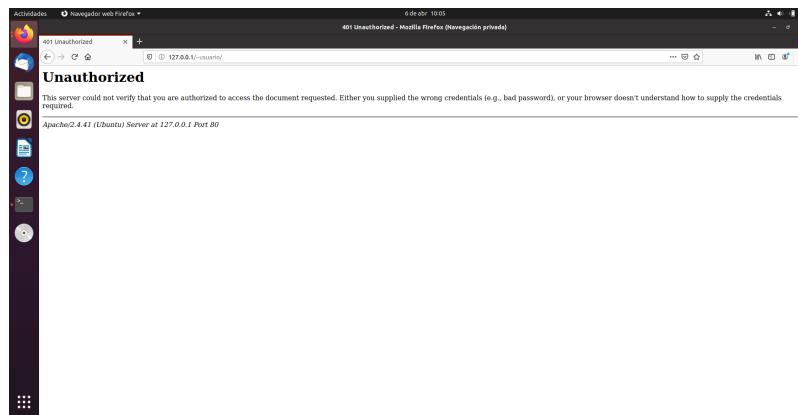


[Everaldo Coelho \(GNU/GPL\)](#)



Captura de pantalla de Firefox (Elaboración propia)

- ✓ La información de autenticación del usuario se envía al servidor, que la verifica y decide si permite o no el acceso al recurso solicitado. Esta información se mantiene en el navegador para utilizarse en posteriores peticiones a ese servidor.
- ✓ Si las credenciales son erróneas Apache muestra "Acceso no autorizado" (código 401).



Captura de pantalla de firefox (Elaboración propia)

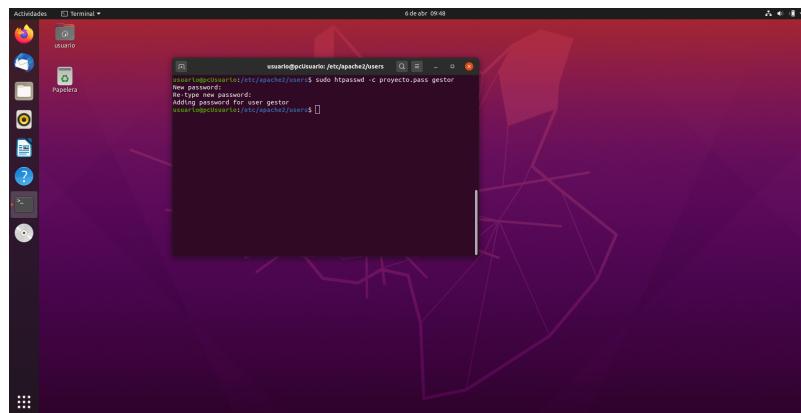
En el servidor web Apache, el que has estado usando en anteriores unidades, existe una utilidad en línea de comandos, htpasswd, que permite almacenar en un fichero una lista de usuarios y sus respectivas contraseñas. La información relativa a las contraseñas se almacena cifrada; aun así, es conveniente crear este fichero en un lugar no accesible por

los usuarios del servidor web. Puedes tener información de este comando y sus opciones escribiendo en la terminal "htpasswd --help" o accediendo a la siguiente web: [Utilidad en línea de comandos, htpasswd](#).

Por ejemplo, para crear el fichero de usuarios "proyecto.pass" y añadirle el usuario "gestor", puedes hacer:

```
sudo htpasswd -c proyecto.pass gestor
```

e introducir la contraseña correspondiente a ese usuario.



Captura de pantalla de Ubuntu (Elaboración propia.)

La opción "-c" indica que se debe crear el fichero, por lo que solo deberás usarla cuando introduzcas el primer usuario y contraseña. Fíjate que en el ejemplo anterior, el fichero "proyecto.pass" se crea en la ruta "/etc/apache2/users", que en principio no es accesible vía web.

Para indicarle al servidor Apache qué recursos tienen acceso restringido, una opción es crear un fichero .htaccess en el directorio en que se encuentren, con las siguientes directivas:

```
AuthName "Contenido restringido"
AuthType Basic
AuthUserFile /etc/apache2/users/proyecto.pass
require valid-user
```

El significado de cada una de las directivas anteriores es el siguiente:

Directivas de autenticación en Apache.

Directiva	Significado
AuthName	Nombre de dominio que se usará en la autenticación. Si el cliente se autentifica correctamente, esa misma información de autenticación se utilizará automáticamente en el resto de las páginas del mismo dominio.
AuthType	Método de autenticación que se usará. Además del método Basic, Apache también permite utilizar el método Digest.

Directiva	Significado
AuthUserFile	Ruta al archivo de credenciales que has creado con <code>htpasswd</code> .
Require	Permite indicar que sólo puedan acceder algunos usuarios o grupos de usuarios concretos. Si indicamos "valid-user", podrán acceder todos los usuarios que se autentifiquen correctamente.

Además tendrás que asegurarte de que en la configuración de Apache se utiliza la directiva `AllowOverride` para que se aplique correctamente la configuración que figura en los ficheros `.htaccess`.

[Directiva `AllowOverride`.](#)

Autoevaluación

La sentencia "sudo htpasswd -c users admin" añade un nuevo usuario con nombre "admin" al fichero "users" que hemos creado anteriormente.

- Verdadero.
- Falso.

Recuerda, ¿cuál es el significado de la opción "`-c`" dentro de la orden anterior?

Efectivamente. Al incluir la opción "`-c`" lo que hacemos es crear un nuevo fichero, con lo cual eliminamos el contenido anterior del mismo.

Solución

1. Incorrecto
2. Opción correcta

1.2.- Mecanismos de autenticación (II).

Desde PHP puedes acceder a la información de autenticación HTTP que ha introducido el usuario utilizando el array "superglobal" `$_SERVER`. Los valores se muestran en la tabla adjunta.

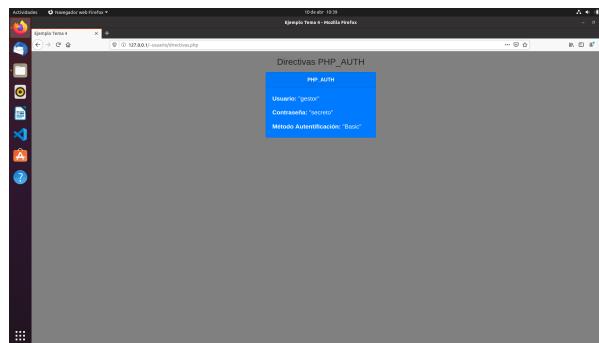


Everaldo Coelho ([GNU/GPL](#))

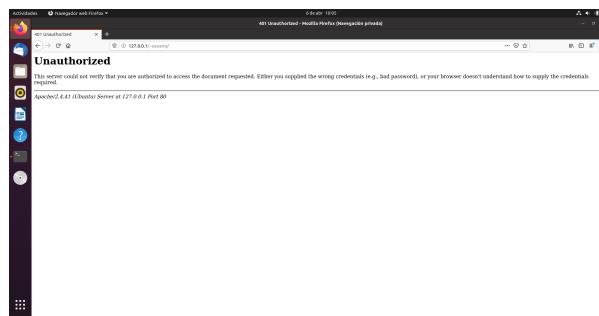
Valores del array `$_SERVER` relacionados con la autenticación

Valor	Contenido
<u><code>\$_SERVER['PHP_AUTH_USER']</code></u>	Nombre de usuario que se ha introducido.
<u><code>\$_SERVER['PHP_AUTH_PW']</code></u>	Contraseña introducida.
<u><code>\$_SERVER['AUTH_TYPE']</code></u>	Método <u>HTTP</u> usado para autenticar. Puede ser <u>Basic</u> O <u>Digest</u> .

Es decir, que si creas una página web que muestre los valores de estas variables, y preparas el servidor web para utilizar autenticación HTTP, cuando accedas a esa página con el usuario "**gestor**" y contraseña "**secreto**" podrás observar los datos de la imagen izquierda de abajo.. Si no introduces un usuario/contraseña válidos, el navegador te mostrará el error **401**.



Captura de pantalla Firefox (Elaboración propia)



Captura de pantalla Firefox (Elaboración propia)

Puedes ver el código de la página anterior en el enlace siguiente: [Código de ejemplo](#) (pdf - 33,94 KB).

Además, en PHP puedes usar la función `header` para forzar a que el servidor envíe un error de "Acceso no autorizado" (código **401**). De esta forma no es necesario utilizar ficheros `.htaccess` para indicarle a Apache qué recursos están restringidos. En su lugar, puedes añadir las siguientes líneas en tus páginas:

```
<?php  
if (!isset($_SERVER['PHP_AUTH_USER'])) {  
    header('WWW-Authenticate: Basic Realm="Contenido restringido"');  
    header('HTTP/1.0 401 Unauthorized');  
    echo "Usuario no reconocido!";  
    exit;  
}  
?>
```

Debes conocer

La función `header` envía las cabeceras HTTP, pero debe utilizarse antes de que se muestre nada por pantalla. En caso contrario, obtendrás un error.

[Manual online sobre la función `header`.](#)

Con el código anterior, la página envía un error **401**, lo que fuerza al navegador a solicitar las credenciales de acceso (nombre de usuario y contraseña). Si se introducen, se ejecuta el resto de la página y se muestra su contenido. En este caso, **habría que añadir algún código para comprobar que el nombre de usuario y contraseña son válidos**. Si se pulsa el botón "Cancelar", se muestra el mensaje de error que se indica.

Ejercicio resuelto

Modifica la página anterior utilizando la función `header` para que solicite las credenciales al usuario, comprueba que el usuario sea "gestor" y la contraseña "secreto".

[Mostrar retroalimentación](#)

Tendrás que crear una página similar a la anterior, y añadir el código para forzar el error 401 antes de cualquier otro.

[Solución de ejemplo.](#) (pdf - 35,91 KB)

1.3.- Incorporación de métodos de autenticación a una aplicación web.

Si utilizas la función `header` para forzar al navegador a solicitar credenciales HTTP, el usuario introducirá un nombre y una contraseña. Pero el servidor no verificará esta información; deberás ser tú quien provea un método para comprobar que las credenciales que ha introducido el usuario son correctas.

El método más simple es incluir en el código PHP de tu página las sentencias necesarias para comparar los datos introducidos con otros datos fijos. Por ejemplo, para permitir el acceso a un usuario "gestor" con contraseña "secreto", puedes hacer:

```
<?php
if ($_SERVER['PHP_AUTH_USER']!='gestor' || $_SERVER['PHP_AUTH_PW']!='secreto') {
    header('WWW-Authenticate: Basic Realm="Contenido restringido"');
    header('HTTP/1.0 401 Unauthorized');
    echo "Usuario no reconocido!";
    exit;
}
?>
```



Spayder26 (CC BY-SA)

Recuerda que el código PHP no se envía al navegador, por lo que la información de autenticación que introduzcas en el código no será visible por el usuario. Sin embargo, esto hará tu código menos portable. Si necesitas modificar el nombre de usuario o la contraseña, tendrás que hacer modificaciones en el código. Además, no podrás permitir al usuario introducir su propia contraseña.

Una solución mejor es utilizar un almacenamiento externo para los nombres de usuario y sus contraseñas. Para esto podrías emplear un fichero de texto, o mejor aún, una base de datos. La información de autenticación podrá estar aislada en su propia base de datos, o compartir espacio de almacenamiento con los datos que utilice tu aplicación web.

Si quieras almacenar la información de los usuarios en la base de datos "proyecto", tienes que crear una nueva tabla en su estructura. Para ello, revisa y ejecuta estas sentencias SQL. Para ello puedes usar `phpMyAdmin` o introducir el comando "`sudo mysql -u root < archivo.sql`". Donde archivo debe tener las sentencias siguientes:

[Sentencias SQL.](#) (pdf - 18,90 KB) (1.00 kB)

Aunque se podrían almacenar las contraseñas en texto plano, por privacidad y seguridad es mejor hacerlo en formato encriptado. En el ejemplo anterior, para el usuario "admin" se almacena el hash (con el algoritmo sha256) correspondiente a la contraseña "secreto".

En PHP puedes usar la función `hash`, por ejemplo `hash('sha256', $cadena)` devuelve el hash utilizando el algoritmo sha256.

En **MySQL** `select sha2('secreto', 256)` nos devolverá el **hash** (utilizando el mismo algoritmo `sha256`) de secreto.

Hay varios algoritmos para calcular el **hash** de una cadena. En nuestro proyecto debemos asegurarnos de usar el mismo tanto **PHP** como en **MySQL**. Por seguridad se recomienda no usar un algoritmo muy usado hasta ahora el `md5`. Encontrarás mas información en el enlace siguiente:

[Función hash.](#)

Ejercicio resuelto

Utiliza la extensión **PDO** para modificar el ejercicio anterior, de tal forma que las credenciales del usuario se comprueben con la información de la nueva tabla "usuarios" creada en la base de datos "proyecto". Si no existe el usuario, o la contraseña es incorrecta, vuelve a pedir las credenciales al usuario.

[Mostrar retroalimentación](#)

Revisa la solución propuesta. Fíjate que se debe usar la función **hash** para comprobar la contraseña. Si introduces un usuario o contraseña incorrectos, el comportamiento depende del navegador que utilices; algunos te pedirán las credenciales de forma indefinida, y otros un número limitado de veces.

[Solución propuesta.](#) (pdf - 27,68 KB)

Autoevaluación

Las dos posibilidades que hemos visto para solicitar al usuario que se autentifique vía **HTTP** son la creación del fichero `.htaccess`, y la utilización de la función `header` de **PHP**. ¿Cuál de esas dos formas será preferible si quieres que los privilegios de acceso a tu aplicación varíen en función del día de la semana (por ejemplo, unos usuarios que puedan acceder de lunes a viernes y otros distintos el fin de semana)?

- El fichero `.htaccess`.
- La función `header`.

Revisa la forma en que se definían los usuarios con acceso al utilizar el fichero `.htaccess`.

Si utilizas la función `header` para enviar los encabezados HTTP, debes escribir tú el código que decide si el usuario se ha autenticado correctamente y se le permite acceder o no. Por tanto, puedes incluir las reglas adicionales que quieras, por ejemplo, darle o no acceso en función del día de la semana.

Solución

1. Incorrecto
2. Opción correcta

2.- Cookies.

Caso práctico

Una vez resuelto el tema de la autenticación, el siguiente objetivo de **Carlos** es aprender a gestionar las **cookies**. **Juan** le ha explicado qué son y cómo funcionan, y seguramente las tengan que utilizar en la aplicación que están desarrollando.



Sabe que muchos de los sitios web que visita las utilizan, porque ha probado a desactivarlas en su navegador, y le han empezado a aparecer mensajes pidiéndole que las vuelva a activar. Se ha propuesto averiguar para qué las utilizan, y cómo las podrán gestionar desde PHP.

Una **cookie** es un fichero de texto que un sitio web guarda en el entorno del usuario del navegador. Su uso más típico es el almacenamiento de las preferencias del usuario (por ejemplo, el idioma en que se deben mostrar las páginas), para que no tenga que volver a indicarlas la próxima vez que visite el sitio.

En PHP, para almacenar una cookie en el navegador del usuario, puedes utilizar la función setcookie. El único parámetro obligatorio que tienes que usar es el nombre de la cookie, pero admite varios parámetros más opcionales. Puedes encontrar más información en el siguiente enlace:

[Función setcookie.](#)

Por ejemplo, si quieras almacenar en una **cookie** el nombre de usuario que se transmitió en las credenciales HTTP (es solo un ejemplo, no es en absoluto aconsejable almacenar información relativa a la seguridad en las **cookies**), puedes hacer:

```
setcookie("nombre_usuario", $_SERVER['PHP_AUTH_USER'], time() + 3600);
```

Los dos primeros parámetros son el nombre de la **cookie** y su valor. El tercero es la fecha de caducidad de la misma (una hora desde el momento en que se ejecute). En caso de no figurar este parámetro, la **cookie** se eliminará cuando se cierre el navegador. Ten en cuenta que también se pueden aplicar restricciones a las páginas del sitio que pueden acceder a una **cookie** en función de la ruta.

Las **cookies** se transmiten entre el navegador y el servidor web de la misma forma que las credenciales que acabas de ver; utilizando los encabezados del protocolo HTTP. Por ello, las sentencias setcookie deben enviarse antes de que el navegador muestre información alguna en pantalla.

El proceso de recuperación de la información que almacena una cookie es muy simple. Cuando accedes a un sitio web, el navegador le envía de forma automática todo el contenido de las cookies que almacene relativas a ese sitio en concreto. Desde PHP puedes acceder a esta información por medio del array `$_COOKIE`.

Siempre que utilices cookies en una aplicación web, debes tener en cuenta que en última instancia su disponibilidad está controlada por el cliente. Por ejemplo, algunos usuarios deshabilitan las cookies en el navegador porque piensan que la información que almacenan puede suponer un potencial problema de seguridad. O la información que almacenan puede llegar a perderse porque el usuario decide formatear el equipo o simplemente eliminarlas de su sistema.

Recomendación

Si una vez almacenada una cookie en el navegador quieras eliminarla antes de que expire, puedes utilizar la misma función `setcookie` pero indicando una fecha de caducidad anterior a la actual.

Ejercicio resuelto

Sobre el mismo ejercicio anterior, almacena en una cookie el último instante en que el usuario visitó la página. Si es su primera visita, muestra un mensaje de bienvenida. En caso contrario, muestra la fecha y hora de su anterior visita. La cookie se guardará una semana.

[Mostrar retroalimentación](#)

Revisa la solución propuesta. Deberás utilizar la función `setcookie` para guardar el instante de su anterior visita y mostrar su contenido utilizando el array `$_COOKIE`.

[Solución propuesta. \(pdf - 43,20 KB\)](#)

Autoevaluación

¿Cuál es la duración por defecto de una cookie si no se indica la fecha de caducidad, como en la siguiente llamada a la función `setcookie`?

```
setcookie("idioma", "español");
```

- Hasta que se cierre el navegador del usuario.
- 1 hora.

Efectivamente, cuando se acaba la sesión del usuario, al cerrar el navegador, se borra la cookie.

Lee bien el texto que habla de los parámetros de la función `setcookie`.

Solución

1. Opción correcta
2. Incorrecto

3.- Manejo de sesiones.

Caso práctico

Carlos ha aprendido a utilizar las **cookies**, y cuáles son sus limitaciones. Está claro que les serán de utilidad, pero también que no cubren las necesidades de una aplicación web. Al viajar en las cabeceras HTTP, la información que pueden mantener está muy limitada.

Aunque sabe que aún le queda mucho por aprender, **Juan** le ha comentado que con los conocimientos que acaba de adquirir ya tiene una buena base para dar los primeros pasos programando en PHP. Dice que sólo le falta aprender cómo funcionan las sesiones para empezar a desarrollar aplicaciones web completas por su cuenta.



Animado por estos comentarios, **Carlos** decide dar un paso más. Veamos cómo funcionan las sesiones en PHP.

Como acabas de ver, una forma para guardar información particular de cada usuario es utilizar **cookies**. Sin embargo, existen diversos problemas asociados a las **cookies**, como el número de ellas que admite el navegador, o su tamaño máximo. Para solventar estos inconvenientes, existen las sesiones. El término **sesión** hace referencia al conjunto de información relativa a un usuario concreto. Esta información puede ser tan simple como el nombre del propio usuario, o más compleja, como los artículos que ha depositado en la cesta de compra de una tienda online.

Cada usuario distinto de un sitio web tiene su propia información de sesión. Para distinguir una sesión de otra se usan los identificadores de sesión (SID). Un SID es un atributo que se asigna a cada uno de los visitantes de un sitio web y lo identifica. De esta forma, si el servidor web conoce el SID de un usuario, puede relacionarlo con toda la información que posee sobre él, que se mantiene en la sesión del usuario. Esta información se almacena en el servidor web, generalmente en ficheros aunque también se pueden utilizar otros mecanismos de almacenamiento como bases de datos.

Como ya habrás supuesto, la cuestión ahora es: ¿y dónde se almacena ese SID, el identificador de la sesión, que es único para cada usuario? Pues existen dos maneras de mantenerlo entre las páginas de un sitio web que visita el usuario:

- ✓ Utilizando **cookies**, tal y como ya viste.
- ✓ Propagándolo en un parámetro de la URL. El SID se añade como una parte más de la misma, de la forma:

<http://www.misitioweb.com/tienda/listado.php&PHPSESSID=34534fg4ffg34ty>

En el ejemplo anterior, el SID es el valor del parámetro **PHPSESSID**.

Ninguna de las dos maneras es perfecta. Ya sabes los problemas que tiene la utilización de cookies. Pese a ello, es el mejor método y el más utilizado. Propagar el SID como parte de la URL conlleva mayores desventajas, como la imposibilidad de mantener el SID entre distintas sesiones, o el hecho de que compartir la URL con otra persona implica compartir también el identificador de sesión.

La buena noticia, es que el proceso de manejo de sesiones en PHP está automatizado en gran medida. Cuando un usuario visita un sitio web, no es necesario programar un procedimiento para ver si existe un SID previo y cargar los datos asociados con el mismo. Tampoco tienes que utilizar la función `setcookie` si quieras almacenar los SID en cookies, o ir pasando el SID entre las páginas web de tu sitio si te decides por propagarlo. Todo esto PHP lo hace automáticamente.

A la información que se almacena en la sesión de un usuario también se le conoce como cookies del lado del servidor (server side cookies). Debes tener en cuenta que aunque esta información no viaja entre el cliente y el servidor, sí lo hace el SID, bien como parte de la URL o en un encabezado HTTP si se guarda en una cookie. En ambos casos, esto plantea un posible problema de seguridad. El SID puede ser conseguido por otra persona, y a partir del mismo obtener la información de la sesión del usuario. La manera más segura de utilizar sesiones es almacenando los SID en cookies y utilizar HTTPS para encriptar la información que se transmite entre el servidor web y el cliente. Algunos consejos sobre sesiones y seguridad los puedes encontrar en el documento web siguiente.

[Manual Sesiones y Seguridad \(PHP\).](#)

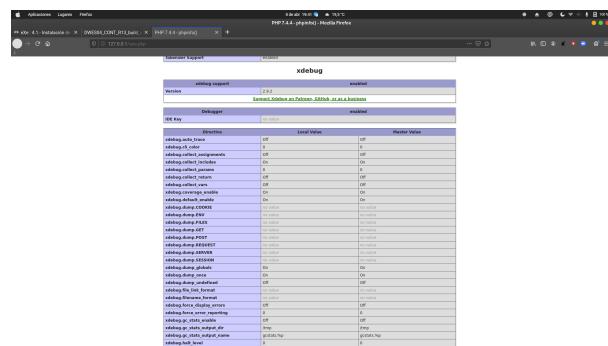
3.1.- Configuración.

Por defecto, PHP incluye soporte de sesiones incorporado. Sin embargo, antes de utilizar sesiones en tu sitio web, debes configurar correctamente PHP utilizando las siguientes directivas en el fichero php.ini según corresponda:

Directivas de configuración de PHP relacionadas con el manejo de sesiones.

Directiva	Significado
session.use_cookies	Indica si se deben usar cookies (1) o propagación en la URL (0) para almacenar el SID.
session.use_only_cookies	Se debe activar (1) cuando utilizas cookies para almacenar los SID, y además no quieras que se reconozcan los SID que se puedan pasar como parte de la URL (este método se puede usar para usurpar el identificador de otro usuario).
session.save_handler	Se utiliza para indicar a PHP cómo debe almacenar los datos de la sesión del usuario. Existen cuatro opciones: en ficheros (files), en memoria (mm), en una base de datos SQLite (sqlite) o utilizando para ello funciones que debe definir el programador (user). El valor por defecto (files) funcionará sin problemas en la mayoría de los casos.
session.name	Determina el nombre de la cookie que se utilizará para guardar el SID. Su valor por defecto es PHPSESSID.
session.auto_start	Su valor por defecto es 0, y en este caso deberás usar la función session_start para gestionar el inicio de las sesiones. Si usas sesiones en el sitio web, puede ser buena idea cambiar su valor a 1 para que PHP active de forma automática el manejo de sesiones.
session.cookie_lifetime	Si utilizas la URL para propagar el SID, éste se perderá cuando cierres tu navegador. Sin embargo, si utilizas cookies, el SID se mantendrá mientras no se destruya la cookie. En su valor por defecto (0), las cookies se destruyen cuando se cierra el navegador. Si quieras que se mantenga el SID durante más tiempo, debes indicar en esta directiva ese tiempo en segundos.
session.gc_maxlifetime	Indica el tiempo en segundos que se debe mantener activa la sesión, aunque no haya ninguna actividad por parte del usuario. Su valor por defecto es 1440. Es decir, pasados 24 minutos desde la última actividad por parte del usuario, se cierra su sesión automáticamente.

La función `phpinfo`, de la que ya hablamos con anterioridad, te ofrece información sobre la configuración actual de las directivas de sesión.



Captura de pantalla Firefox (Elaboración propia)

Para saber más

En la documentación de [PHP](#) tienes información sobre éstas y otras directivas que permiten configurar el manejo de sesiones.

Directivas que permiten configurar el manejo de sesiones.

Autoevaluación

Si la información del usuario que quieras almacenar incluye contenido privado como una contraseña, ¿qué utilizarías, cookies o la sesión del usuario?

- La sesión del usuario.
 - Cookies.

La sesión del usuario se almacena y se procesa en el servidor web, por lo que no es necesario transmitirla hasta el ordenador del usuario, lo que aumenta su seguridad.

Las **cookies** deben transmitirse entre el navegador del usuario, que es dónde se almacenan, hasta el servidor web para ser procesadas.

Solución

1. Opción correcta
2. Incorrecto

3.2.- Inicio y fin de una sesión.

El inicio de una sesión puede tener lugar de dos formas. Si has activado la directiva `session.auto_start` en la configuración de PHP, la sesión comenzará automáticamente en cuanto un usuario se conecte a tu sitio web. En caso de que ese usuario ya haya abierto una sesión con anterioridad, y esta no se haya eliminado, en lugar de abrir una nueva sesión se reanudará la anterior. Para ello se utilizará el SID anterior, que estará almacenado en una **cookie** (recuerda que si usas propagación del SID, no podrás restaurar sesiones anteriores; el SID figura en la URL y se pierde cuando cierras el navegador).

Si por el contrario, decides no utilizar el inicio automático de sesiones, deberás ejecutar la función `session_start` para indicar a PHP que inicie una nueva sesión o reanude la anterior. Aunque anteriormente esta función devolvía siempre `true`, a partir de la versión **5.3.0** de PHP su comportamiento es más coherente: devuelve `false` en caso de no poder iniciar o restaurar la sesión.



[Everaldo Coelho and YellowIcon \(GNU/GPL\)](#)

Como el inicio de sesión requiere utilizar cookies, y éstas se transmiten en los encabezados HTTP, debes tener en cuenta que para poder iniciar una sesión utilizando `session_start`, tendrás que hacer las llamadas a esta función antes de que la página web muestre información en el navegador.

Además, todas las páginas que necesiten utilizar la información almacenada en la sesión, deberán ejecutar la función `session_start`.

Mientras la sesión permanece abierta, puedes utilizar la variable superglobal `$_SESSION` para añadir información a la sesión del usuario, o para acceder a la información almacenada en la sesión. Por ejemplo, para contar el número de veces que el usuario visita la página, puedes hacer:

```
<?php
// Iniciamos la sesión o recuperamos la anterior sesión existente
session_start();
// Comprobamos si la variable ya existe
if (isset($_SESSION['visitas']))
    $_SESSION['visitas']++;
else
    $_SESSION['visitas'] = 0;
?>
```

Si en lugar del número de visitas, quisieras almacenar el instante en que se produce cada una, la variable de sesión "visitas" deberá ser un array y por tanto tendrás que cambiar el código anterior por:

```
<?php
// Iniciamos la sesión o recuperamos la anterior sesión existente
```

```
session_start();  
// En cada visita añadimos un valor al array "visitas"  
$_SESSION['visitas'][] = mktime();  
?>
```

Aunque como ya viste, puedes configurar PHP para que elimine de forma automática los datos de una sesión pasado cierto tiempo, en ocasiones puede ser necesario cerrarla de forma manual en un momento determinado. Por ejemplo, si utilizas sesiones para recordar la información de autenticación, deberás darle al usuario del sitio web la posibilidad de cerrar la sesión cuando lo crea conveniente.

En PHP tienes dos funciones para eliminar la información almacenada en la sesión:

- ✓ session_unset. Elimina las variables almacenadas en la sesión actual, pero no elimina la información de la sesión del dispositivo de almacenamiento usado.
- ✓ session_destroy. Elimina completamente la información de la sesión del dispositivo de almacenamiento.

Ejercicio resuelto

Crea una página similar a la anterior, almacenando en la sesión de usuario los instantes de todas sus últimas visitas. Si es su primera visita, muestra un mensaje de bienvenida. En caso contrario, muestra la fecha y hora de todas sus visitas anteriores. Añade un botón a la página que permita borrar el registro de visitas.

Utiliza también una variable de sesión para comprobar si el usuario se ha autenticado correctamente. De esta forma no hará falta comprobar las credenciales con la base de datos constantemente.

[Mostrar retroalimentación](#)

Revisa la solución propuesta. Acuérdate de que debes hacer una llamada a la función session_start() antes de utilizar la variable superglobal \$_SESSION para acceder a la información de la sesión.

[Solución propuesta. \(pdf - 45,44 KB\)](#)

Autoevaluación

Si usamos el inicio de sesión automático, la sesión de un usuario se inicia en cuanto se autentifica correctamente en el servidor web.

- Falso.
- Verdadero.

Con el inicio de sesión automático, la sesión del usuario se inicia en cuanto se conecta al sitio web, esté o no autenticado. No tiene nada que ver una cosa con otra. Un usuario no autenticado puede tener una sesión de la misma forma que otro que sí se haya autenticado correctamente.

Y si un sitio web no requiere que sus usuarios se autentifiquen (es de libre acceso), ¿no puede usar sesiones?

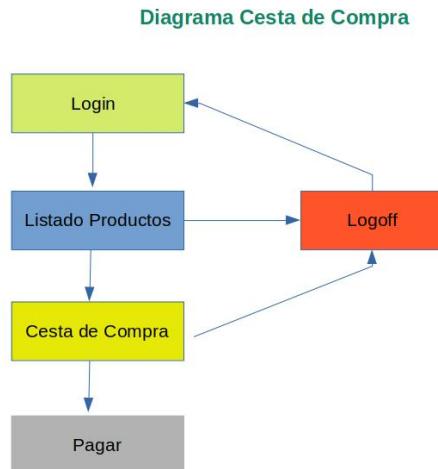
Solución

1. Opción correcta
2. Incorrecto

3.3.- Gestión de la información de la sesión (I).

En este punto vas a ver paso a paso un ejemplo de utilización de sesiones para almacenar la información del usuario. Utilizarás la base de datos " proyecto ", creada anteriormente, para crear un prototipo de una tienda web dedicada a la venta de productos de informática.

Las páginas de que constará tu tienda online son las siguientes:



Writer y Gimp (Elaboración propia.)

- ✓ **conexion.php.** En ella meteremos los parámetros para iniciar la conexión y alguna otra función útil. Con `require_once` la "llamaremos" desde los otros archivos, lo que nos evitirá tener que repetir código
- ✓ **login.php.** Su función es autenticar al usuario de la aplicación web. Todos los usuarios de la aplicación deberán autenticarse utilizando esta página antes de poder acceder al resto de páginas.
- ✓ **listado.php.** Presenta un listado de los productos de la tienda, y permite al usuario seleccionar aquellos que va a comprar.
- ✓ **cesta.php.** Muestra un resumen de los productos escogidos por el usuario para su compra y da acceso a la página de pago.
- ✓ **pagar.php.** Una vez confirmada la compra, la última página debería ser la que permitiera al usuario escoger el método de pago y la forma de envío. En este ejemplo no la vas a implementar como tal. Simplemente mostrará un mensaje de tipo "Gracias por su compra" y ofrecerá un enlace para comenzar una nueva compra.
- ✓ **cerrar.php.** Nos permitirá cerrar la sesión y nos devolverá a la página de "login".

Una vez "logueados" se nos dará la opción de cerrar sesión en todas las páginas

Recuerda poner a las páginas los nombres que aquí figuran, almacenando todas en la misma carpeta. Si cambias algún nombre o mueves alguna página de lugar, los enlaces internos no funcionarán.

Aunque el aspecto de la aplicación no es importante para nuestro objetivo, se ha usado **Bootstrap** y los iconos de **FontAwesome** para mejorar fácilmente y rápidamente el diseño

de la misma. Se ha optado por incorporar los _____CDN, para no tener que descargar los estilos por lo que será necesario internet para que puedas ver los mismos. Para poder usarlos basta con poner lo siguiente en el `head` de las páginas.

```
<!-- css para usar Bootstrap -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap
    integrity="sha384-Vkoo8x4CGs03+Hhxv8T/Q5PaXtkKtu6ug5T0eNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
<!-- css Fontawesome -->
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.3.1/css/all.css"
    integrity="sha384-mzrmE5qonljUremFsqc01SB46JvROS7bZs3IO2EmfFsd15uHvIt+Y8vEf7N7fWAU"
```



Se recomienda también usar las extensiones adecuadas en VSC para poder trabajar con estos estilos. Poniendo **Bootstrap** en el gestor de extensiones te aparecen varias, prueba hasta que encuentres una que te guste.

Antes de comenzar ten en cuenta que la aplicación que vas a desarrollar no es completamente funcional. Además de no desarrollar la página con la información de pago, habrá algunas opciones que no tendrás en cuenta para simplificar el código. Por ejemplo:

- ✓ No tendrás en cuenta la posibilidad de que el usuario compre varias unidades de un mismo producto.
- ✓ Una vez añadido un producto a la cesta de compra, no se podrá retirar de la misma. La única posibilidad será vaciar toda la cesta y comenzar de nuevo añadiendo productos.
- ✓ No se mostrarán imágenes de los productos, ni será posible ver el total de la compra hasta que ésta haya finalizado.
- ✓ Se muestran todos los productos en una única página. Sería preferible filtrarlos por familia y mostrarlos en varias páginas, limitando a 10 o 20 productos el número máximo de cada página.

Recomendación

Aunque reduzcamos en este ejemplo la funcionalidad de la tienda, te animamos a que una vez finalizado el mismo, añadas por tu cuenta todas aquellas opciones que quieras. Recuerda que la mejor forma de aprender programación es... ¡programando!

3.4.- Gestión de la información de la sesión (II).

La primera página que vas a programar es la de autenticación del usuario (`login.php`). Para variar, utilizarás las capacidades de manejo de sesiones de PHP para almacenar la identificación de los usuarios. Además, utilizaremos la información de la tabla "**usuarios**" en la base de datos "**proyecto**", accediendo mediante PDO en lugar de **MySQLi**. Para la conexión nos crearemos el archivo "`conexión.php`" al que llamaremos con `require_once` en las páginas que necesitemos. Si has copiado el código para añadir la tabla **usuarios**, habrá en la misma dos usuario que podamos usar, **admin** con contraseña **secreto** y **gestor** con contraseña **pass**.

Vas a crear en la página un formulario con dos campos, uno de tipo `text` para el usuario, y otro de tipo `password` para la contraseña. Al pulsar el botón Enviar, el formulario se enviará a esta misma página, donde se compararán las credenciales proporcionadas por el usuario con las almacenadas en la base de datos. Si los datos son correctos, se iniciará una nueva variable de sesión y se almacenará en ella el nombre del usuario que se acaba de conectar.

Empecemos viendo detenidamente cada una de las páginas:

`conexión.php` : Contendrá el código para hacer la conexión a la base de datos "**proyecto**". Como estamos en desarrollo ponemos los errores en `Exception` y los visualizamos. También las funciones `cerrar()` y `cerrarTodo()` para cerrar las conexiones. Fíjate que pasamos las variables por referencia.

`login.php` : Aunque ahora, los estilos no son demasiados importantes y, nos centraremos más en el código PHP, usando Bootstrap y los iconos de fontawesome será muy sencillo hacer un formulario profesional. Hay muchos ejemplos en la web, que se pueden usar, de formularios de `login` con Bootstrap. En el apartado anterior se explica como poder usar estos recursos.

Veamos este archivo detenidamente:

- ✓ Incluimos el fichero "`conexion.php`" , después se crea una función de error a la que pasaremos un mensaje de error, inicializamos la variable `$_SESSION['error']` con dicho mensaje y volvemos a cargar la página.

```
<?php
session_start();
require_once 'conexion.php';
function error($mensaje)
{
    $_SESSION['error'] = $mensaje;
    header('Location:login.php');
```

Recorte captura de pantalla (Elaboración propia)

```
die();
}
```

- ✓ En el `body`, comprobamos si hemos enviado el formulario, si es así hacemos una breve comprobación de que los campos contienen algún carácter (el atributo `required` del formulario nos permite enviar solo espacios en blanco, lo cual puede ser un problema) que no sea un espacio en blanco, calculamos el `hash` (utilizando `sha256` que es como almacenamos las contraseñas en la base de datos) a la contraseña y hacemos la consulta como ya sabemos.

```
$nombre = trim($_POST['usuario']);
$pass = trim($_POST['pass']);
if (strlen($nombre) == 0 || strlen($pass) == 0) {
    error("Error, El nombre o la contraseña no pueden contener solo espacios en blanco");
}

//Creamos el sha256 de la contraseña que es como se almacena en mysql

$pass1 = hash('sha256', $pass);
$consulta = "select * from usuarios where usuario=:u AND pass=:p";
$stmt = $conProyecto->prepare($consulta);
try {
    $stmt->execute([
        ':u' => $nombre,
        ':p' => $pass1
    ]);
} catch (PDOException $ex) {
    cerrarTodo($conProyecto, $stmt);
    error("Error en la consulta a la base de datos.");
}
```

- ✓ Comprobamos el resultado de la consulta.

Si la misma devuelve **0** filas es que las credenciales son invalidas, mostramos un error y volvemos a cargar la página, si no, inicializamos la variable de sesión `$_SESSION['nombre']` con el nombre del usuario y nos vamos a la página 'listado.php'

```
if ($stmt->rowCount()==0) {
    cerrarTodo($conProyecto, $stmt);
    error("Error, Nombre de usuario o password incorrecto");
}
cerrarTodo($conProyecto, $stmt);

//Nos hemos validado correctamente creamos la sesion de usuario con el nombre de usuario

$_SESSION['nombre'] = $nombre;
header('Location:listado.php');
```

Si **no** hemos enviado el formulario mostramos el mismo y después de él mostramos los errores (si los hay), acuérdate que los estamos guardando en la variable

`$_SESSION['error']`. Una vez mostrados si los hay hacemos `unset` de dicha variable para no mostrar el error cada vez que carguemos 'login.php'

```
if (isset($_SESSION['error'])) {  
    echo "<div class='mt-3 text-danger font-weight-bold text-lg'>";  
    echo $_SESSION['error'];  
    unset($_SESSION['error']);  
    echo "</div>";  
}
```

Debajo tienes el código de ambas páginas

- ✓ Descargar "conexion.php" : [conexión](#) (pdf - 30,28 KB)
- ✓ Descargar "login.php" : [login](#) (pdf - 41,91 KB)

Autoevaluación

En el código anterior, la sesión del usuario se inicia solo si proporciona un nombre de usuario y contraseña correctos. ¿Se podría haber iniciado la sesión al principio del código, aunque el usuario no proporcione credenciales?

- No.
- Sí.

Recuerda lo que comentábamos anteriormente acerca de la sesión de los usuarios y la autenticación de los mismos.

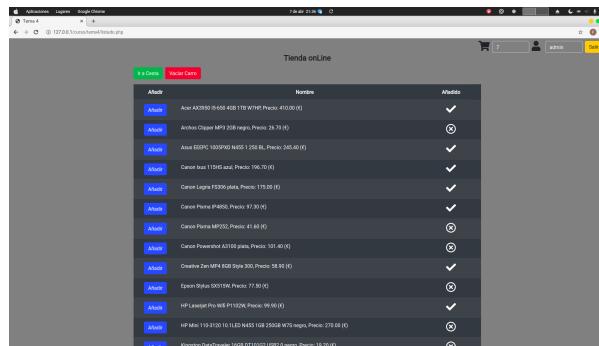
Sí, se puede iniciar la sesión de igual forma, aunque si el usuario no se autentifica no tendríamos ninguna información que incluir dentro de la misma.

Solución

1. Incorrecto
2. Opción correcta

3.5.- Gestión de la información de la sesión (III).

Cuando un usuario proporciona unas credenciales de inicio de sesión correctas (recuerda que tú ya habías añadido el usuario "**admin**" con contraseña "**secreto**"), se le redirige de forma automática a la página del listado de productos ("listado.php") para que pueda empezar a hacer la compra. Esa es la página que vas a programar a continuación. Como ya se indicó en el apartado anterior se hace uso para los estilos de **Bootstrap** Y **Font Awesome**.



Captura de pantalla. (Elaboración propia)

La página se divide en varias partes:

- ✓ Al principio comprobamos si existe la variable `$_SESSION['nombre']`, la que creamos al hacer el `login`, si no es así volvemos a "login.php", de esta forma garantizamos que nadie, sin validar, pueda acceder a esta página aunque conozca su URL y la ponga en el navegador.

```
<?php
session_start();
if (!isset($_SESSION['nombre'])) {
    header('Location:login.php');
}
require_once 'conexion.php';
$consulta = "select id, nombre, pvp from productos order by nombre";
$stmt = $conProyecto->prepare($consulta);
try {
    $stmt->execute();
} catch (PDOException $ex) {
    cerrarTodo($conProyecto, $stmt);
    die("Error al recuperar los productos " . $ex->getMessage());
}
?>
```

después hacemos el `require_once` a "conexion.php", hacemos una consulta para recuperar el **id**, **nombre** y **pvp** de todos los productos, que mostraremos en la tabla con el listado de productos.

- ✓ **encabezado**. Contiene la cantidad de elementos del carro (para ello, simplemente contamos los elementos del array `$_SESSION['cesta']`), el nombre del usuario (este lo

mandamos en la variable `$_SESSION['nombre']`) y un botón para cerrar sesión, este botón nos redirige a la página "cerrar.php".

```
<div class="float float-right d-inline-flex mt-2">
    <i class="fa fa-shopping-cart mr-2 fa-2x"></i>
    <?php
        if (isset($_SESSION['cesta'])) {
            $cantidad = count($_SESSION['cesta']);
            echo "<input type='text' disabled class='form-control mr-2 bg-transparent text-
        } else {
            echo "<input type='text' disabled class='form-control mr-2 bg-transparent text-
        }

    ?>
    <i class="fas fa-user mr-3 fa-2x"></i>
    <input type="text" size='10px' value=<?php echo $_SESSION['nombre']; ?>" class="fo
    mr-2 bg-transparent text-white" disabled>
    <a href="cerrar.php" class="btn btn-warning mr-2">Salir</a>
</div>
```

- ✓ **Botones** "Ir a cesta" y "Vaciar Carro". "Ir a cesta" es un enlace a la página "cesta.php" y "Vaciar Carro" es un submit a "listado.php". La página en la que estamos.

```
<form class="form-inline" name="vaciar" method="POST" action='<?php echo $_SERVER['PHP_
    <a href="cesta.php" class="btn btn-success mr-2">Ir a Cesta</a>
    <input type="submit" value='Vaciar Carro' class="btn btn-danger" name="vaciar">
</form>
```

Al cargar la página comprobaremos si hemos enviado el formulario anterior de nombre "**vaciar**", si es así hacemos `unset()` de la variable `$_SESSION['cesta']`

```
if (isset($_POST['vaciar'])) {
    unset($_SESSION['cesta']);
}
```

- ✓ **productos**. Contiene el listado de todos los productos ordenados por nombre tal y como figuran en la base de datos. Cada producto figura en una fila (nombre y precio). Se crea un formulario, por cada producto, con un botón "**Añadir**" que envía a esta misma página el **id** del producto añadido. Cuando se abre la página, se comprueba si se ha enviado este formulario, y si fuera así se añade un elemento al array asociativo `$_SESSION['cesta']` con el **id** del producto. A la derecha aparece un check (ícono `fa-check`) o una X (ícono `fa-times-circle`) en función de que el producto haya sido añadido o no. Para mostrar los productos hacemos uso de la consulta de arriba.

```
while ($filas = $stmt->fetch(PDO::FETCH_OBJ)) {
    echo "<tr><th scope='row' class='text-center'>";
    echo "<form action='$_SERVER['PHP_SELF']' method='POST'>";
    echo "<input type='hidden' name='id' value='{$filas->id}'>";
```

```
echo "<input type='submit' class='btn btn-primary' name='comprar' value='Añadir al carrito' />";
echo "</form>";
echo "</th>";
echo "<td>{$filas->nombre}, Precio: {$filas->pvp} (€)</td>";
echo "<td class='text-center'>";
if (isset($_SESSION['cesta'][$filas->id])) {
    echo "<i class='fas fa-check fa-2x'></i>";
} else {
    echo "<i class='far fa-times-circle fa-2x'></i>";
}
echo "<td>";
echo "</tr>";

}
cerrarTodo($conProyecto, $stmt);
```

Descargar "listado.php": [listado](#) (pdf - 42,62 KB)

Autoevaluación

Para proteger el acceso a "listado.php" es suficiente que el índice de nuestro sitio sea la página de login y que desde allí, redireccionar a "listado.php" sólo si el login ha sido válido.

Sugerencia

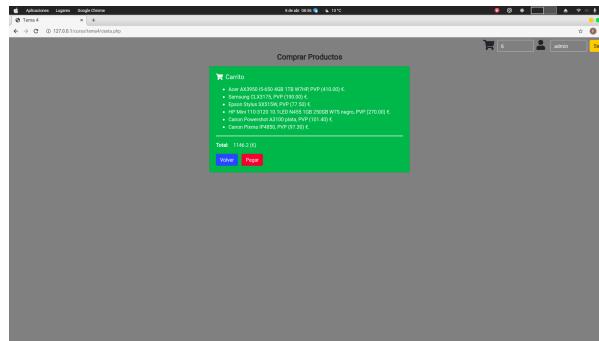
- Verdadero Falso

Falso

Es necesario que al hacer login inicializamos una variable de sesión que luego vayamos comprobando si existe en cada página que queramos limitar el acceso. Si no existe podemos, con header, redireccionar a la página de login.

3.6.- Gestión de la información de la sesión (IV).

Si desde la página del listado de productos, el usuario pulsa sobre el botón "Ir a Cesta", se le dirige a la página de la Cesta de la compra "cesta.php", en la que se le muestra un resumen de los productos que ha seleccionado junto al importe total de los mismos.



Captura de pantalla del navegador. (Elaboración propia.)

La página se divide en varias partes:

- ✓ Al principio, como ya explicamos en el apartado anterior comprobamos que existe la variable `$_SESSION['nombre']`, redireccionándonos a `login` si no es así. Hacemos el `require_once` de `conexión` y recorriendo los valores del array `$_SESSION['cesta']` (recuerda que almacenaba los `id` de los productos seleccionados) inicializamos un array con el nombre y el precio de los mismos. Para ello utilizamos la función `recuperarProducto($id)` de `"conexión.php"`.

```

if(!isset($_SESSION['nombre'])){
    header('Location:login.php');
}
require_once 'conexion.php';
if(isset($_SESSION['cesta'])){
    foreach($_SESSION['cesta'] as $k=>$v){
        $producto=consultarProducto($k);
        $listado[$k]=[$producto->nombre, $producto->pvp];
        $producto=null;
    }
    cerrar($conProyecto);
}

```

- ✓ **Encabezado**, la cantidad de productos del carro, el usuario con el que estamos validados y el botón "**Salir**" es exactamente igual que en la página "`listado.php`".
- ✓ **Listado de productos**, un listado de los productos añadidos (nombre y precio) y la suma de los precios.

```

<?php
if(!isset($_SESSION['cesta'])){

```

```

        echo "<p class='card-text'>Carrito Vacío</p>";
    }
    else{
        $total=0;
        echo "<p class='card-text'>";
        echo "<ul>";
        foreach($listado as $k=>$v){
            echo "<li>$v[0], PVP ($v[1]) €.</li>";
            $total+=$v[1];
        }
        echo "</ul></p>";
        echo "<hr style='border:none; height:2px; background-color: white'>";
        echo "<p class='card-text'><b>Total:</b><span class='ml-3'>$total (€)</span></p>";
    }
?>
```

- ✓ **Botones "Volver" y "Comprar".** Son un enlace a "listado.php" y "pagar.php" respectivamente.

```
<a href="listado.php" class="btn btn-primary mr-2">Volver</a>
<a href="pagar.php" class="btn btn-danger">Pagar</a>
```

La página "pagar.php" (es una página sencilla que nos da la opción a hacer otra compra o a salir).

El contenido de la página "cerrar.php" sería:

```
<?php
session_start();
unset($_SESSION['nombre']);
unset($_SESSION['cesta']);
header('Location:login.php');
```

- ✓ Descargar "cesta.php": [cesta](#) (pdf - 39,87 KB)
- ✓ Descargar "cerrar.php": [cerrar](#) (pdf - 22,43 KB)
- ✓ Descargar "pagar.php": [pagar](#) (pdf - 19,31 KB)
- ✓ Descargar "cesta.zip": [Todo el Proyecto.](#) (zip - 6,74 KB)

Con esto ya estaría terminado este proyecto. Recuerda que puedes introducirle cantidad de mejoras, y que hay múltiples formas de hacer lo mismo.

Citas Para Pensar

“ Somos lo que hacemos repetidamente. La excelencia, entonces, no es un acto sino un hábito.

Aristóteles.

Autoevaluación

En el proyecto anterior un usuario validado tenía acceso a todas las páginas del proyecto ¿Podríamos establecer restricciones de acceso a distintos usuarios?

- Sí.
- No.

Efectivamente, con un nuevo campo en la tabla **usuarios** (por ejemplo **perfil**) podríamos crear otra variable de sesión en "login.php" a parte de `$_SESSION['nombre']` y controlar, en función del valor de esta nueva variable, el acceso a determinadas páginas o partes de ellas.

Revisa el funcionamiento que le hemos dado a las variables de sesión para permitir o no el acceso.

Solución

1. Opción correcta
2. Incorrecto

4.- Herramientas para depuración de código.

Caso práctico

Con lo que ha aprendido, **Carlos** se ha aventurado por su cuenta en la programación de una aplicación web sencilla. Ahora ya sabe cómo y dónde almacenar la información que genera cada página, para poderla utilizar más adelante. Ha cogido la página que había creado para catalogar su colección de comics, y ha decidido convertirla en una aplicación de verdad. Empieza a estructurar el código que tenía y decide programar algunas páginas más y crear un menú para moverse entre ellas.



Y entonces... comienzan a surgir los problemas: páginas que no muestran lo que se suponía que deberían mostrar, errores inexplicables cuando se intenta conectar a la base de datos, listados que algunas veces funcionan y otras no,...

Tras muchas horas intentando encontrar y solucionar los problemas que van apareciendo, busca en Internet y descubre que existen algunas utilidades que pueden ayudarle a arreglar los errores del código. No se trata de que no vaya a cometer fallos nunca más, sino de que cuando los cometa tenga una forma más cómoda y rápida de resolverlos.

Carlos está descubriendo por su cuenta cómo funciona la depuración de código en lenguaje PHP.

Seguramente en la aplicación de tienda online que acabas de programar te hayas encontrado con algunos problemas. Algun código que no funcionaba, problemas de conexión a la base de datos, o páginas que no mostraban lo que deberían.

Con lo que has visto hasta ahora ya puedes empezar a desarrollar algunas aplicaciones web sencillas en lenguaje PHP. Sin embargo, cuando empiezan a surgir problemas tienes que ingeníártelas para poder descubrir qué es lo que está fallando. Puedes poner trozos de código en las páginas que desarrollas para mostrar información interna de la aplicación, que muestren en pantalla la secuencia en que se ejecutan las líneas de código, el contenido de las variables que usas o incluso detener la ejecución del script. Para ello puedes utilizar echo, print, print_r, var_dump, die(\$var) o exit.

Para saber más

En la documentación de PHP puedes consultar información sobre la función [var_dump](#).

[Función var_dump](#)

Por ejemplo, si quieras ver el contenido de la variable superglobal `$_SERVER`, puedes introducir una línea como:

```
<?php  
    var_dump($_SERVER);  
    echo "<br>";  
    print_r($_SERVER);  
?>
```

Este método de depuración es muy tedioso y requiere hacer cambios constantes en el código de la aplicación. También existe la posibilidad de que una vez encontrado el fallo, te olvides de eliminar de tus páginas algunas de las líneas creadas a propósito para la depuración, con los consiguientes problemas.

Si ya has programado anteriormente en otros lenguajes, seguramente conoces algunas herramientas de depuración específicas que se integran con el entorno de desarrollo, permitiéndote por ejemplo detener la ejecución cuando se llega a cierta línea y ver el contenido de las variables en ese momento. Al usar estas herramientas minimizas o eliminas por completo la necesidad de introducir líneas específicas de depuración en tus programas y agilizas el procedimiento de búsqueda de errores.

De las herramientas disponibles que posibilitan la depuración en lenguaje [PHP](#), en esta unidad aprenderás a configurar y utilizar la extensión [Xdebug](#). Es una extensión de código libre, bajo licencia propia ([The Xdebug License](#), basada en la licencia de [PHP](#)) que se integra perfectamente con [VSC](#).

Si la depuración está activada, [Xdebug](#) controla la ejecución de los guiones en [PHP](#). Puede pausar, reanudar y detener los programas en cualquier momento. Cuando el programa está pausado, [Xdebug](#) puede obtener información del estado de ejecución, incluyendo el valor de las variables (puede incluso cambiar su valor).

[Xdebug](#) es un servidor que recibe instrucciones de un cliente (normalmente por el puerto 9000), que en nuestro caso será [VSC](#). De esta forma, no es necesario que el entorno de desarrollo esté en la misma máquina que el servidor con [Xdebug](#).

4.1.- Instalación de herramientas de depuración.

Concretamente, vas a aprender a preparar un sistema Ubuntu para depurar aplicaciones en lenguaje PHP utilizando Xdebug y el IDE Visual Studio Code.

Aunque existen diversas formas para realizar la instalación de Xdebug. Probablemente la más sencilla es a través del repositorio de extensiones PECL, del que ya hablamos en la unidad 2. Necesitas por tanto asegurarte de que PECL está disponible en tu instalación de PHP. Para instalarlo desde los repositorios de Ubuntu, ejecuta desde una consola:



[Devcore](#) (Dominio público)

```
sudo apt-get install php-pear
```

Es conveniente mantener actualizados los repositorios ejecutando `sudo apt-get update` antes de instalar paquetes.

Además, para poder compilar las extensiones que descargas con PECL, debes instalar el paquete con las herramientas de desarrollo `php-dev`. Incluye dos programas necesarios para instalar Xdebug: `phpize` y `php-config`.

```
sudo apt-get install php-dev
```

Si todo fue bien, ya tienes el sistema preparado para instalar Xdebug.

```
sudo pecl install xdebug
```

Al finalizar la instalación verás un mensaje como el siguiente:

```
user@pcUser: ~
=====
INSTALLATION INSTRUCTIONS
=====
See https://xdebug.org/install.php#configure-php for instructions
on how to enable Xdebug for PHP.

Documentation available online as well:
  - List of all settings: https://xdebug.org/docs/settings.php
  - List of functions: https://xdebug.org/docs/functions.php
  - Remote extensions: https://xdebug.org/docs/remote.php
  - Remote debugging: https://xdebug.org/docs/debugger.php

NOTE: Please disregard the message
      You should add "extension=xdebug.so" to php.ini
      that was emitted by the PECL installer. This does not work for
      Xdebug.

Build process completed successfully
install ok channel://pecl.php.net/xdebug-2.9.4
install ok https://pecl.php.net/xdebug-2.9.4
install ok https://pecl.php.net/xdebug-2.9.4/info/location
You should add "zend_extension=/usr/lib/php/20190902/xdebug.so" to php.ini
user@pcUser: ~
```

Captura de pantalla (Elaboración propia)

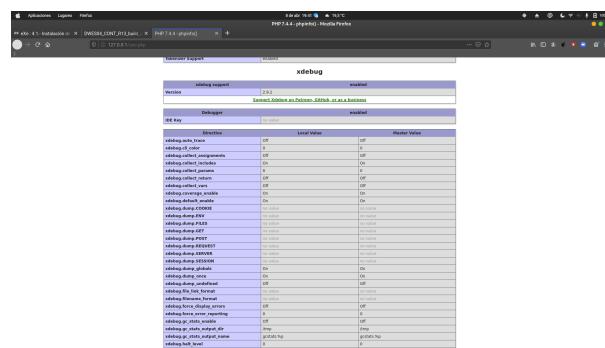
En esa pantalla figura (tras "Installing") la ruta en que se ha instalado la extensión (fichero **xdebug.so**). Además te indica que debes modificar el fichero "**php.ini**" para activarla. Abre por tanto el fichero "**php.ini**" (en la fecha de elaboración de este manual la ruta es **/etc/php/7.4/apache2/php.ini**) de tu instalación de **PHP**, y añade las siguientes líneas al final.

```
[xdebug]
zend_extension="/usr/lib/php/20190902/xdebug.so"
; Opciones de configuración
xdebug.remote_enable = 1
xdebug.remote_autostart = 1
```

Ajusta la ruta anterior al fichero "**xdebug.so**" para que sea la misma que has obtenido en tu sistema tras instalar la extensión.

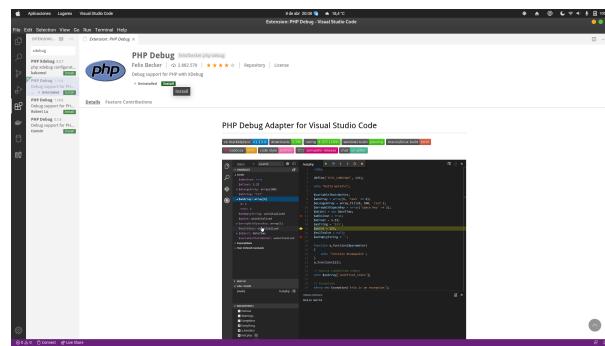
Por último, recuerda reiniciar el servidor web siempre que modifiques el fichero "**php.ini**" para aplicar los cambios.

Al final podemos comprobar con el archivo "**info.php**" en la parte correspondiente a **xdebug** que todo está funcionando.



Captura de pantalla de Firefox. (Elaboración propia.)

Con **Xdebug** instalado y funcionando, preparamos **Visual Studio Code**, este nos ofrece varias extensiones para trabajar con **Xdebug**, el funcionamiento de ellas es muy parecido. Instalaremos la extensión: **PHP Debug** de *"Felix Becker"*



Captura de Pantalla de Visual Studio Code (Elaboracion propia)

Para saber más

Si necesitas instalar `Xdebug` en sistemas Windows, puedes encontrar información al respecto en Internet. En la página web de la extensión `Xdebug` tienes un asistente que analiza tu archivo `phpinfo()` y te recomienda lo que tienes que hacer para la instalación (información en lenguaje inglés).

[Página web de la extensión asistente Xdebug.](#)

4.2.- Depuración de código en PHP.

Entre las características de depuración que incorpora Xdebug destacan:

- ✓ Creación de registros con las llamadas a funciones que se produzcan, incluyendo parámetros y valores devueltos.
- ✓ Creación de registros de optimización, que permitan analizar el rendimiento de los programas.
- ✓ Depuración remota.

Vamos a centrarnos en la depuración remota utilizando VSC con la extensión "PHP Debug" instalada.



[djcowan](#) (Dominio público)

- ✓ **Paso 1:** Abrimos el archivo PHP del que vayamos a hacer el debug (en este caso "cesta.php"). Le damos al Triángulo con el "bug" encima.
- ✓ **Paso 2:** La primera vez nos pide crear un archivo "launch.json" con la configuración. Pulsamos sobre "create launch.json file".
- ✓ **Paso 3:** Al pulsar nos pide el lenguaje del fichero sobre el que vamos a hacer debug, seleccionamos PHP.
- ✓ **Paso 4:** Vemos que nos ha creado un json con una configuración estándar, si no hemos cambiado nada en Xdebug lo dejamos tal cuál.
- ✓ **Paso 5:** Ponemos de ejemplo dos "BreakPoints" en cesta.php.
- ✓ **Paso 6:** Le damos a RUN para empezar la depuración.
- ✓ **Paso 7:** Cargamos el archivo "cesta.php", en el navegador, para llegar a él hacemos login, seleccionamos dos o tres productos de listado.php y pulsamos "Ir a cesta", en este momento vemos que se nos abre VSC. Nos aparece la linea 9 (donde pusimos el primer Breakpoint) de amarillo y arriba tenemos un pequeño panel de botones (continue, step into, step over ...) si observamos a la izquierda ya me aparecen variable y sus valores, en el caso de \$listado aparece uninitialized.
- ✓ **Paso 8:** Como se ha parado en la llamada a consultar producto, le damos a "step into (F11)" y vemos que se nos abre en VSC la página conexión.php y la función consultarProducto(), en el panel de arriba le damos a continuar (play)
- ✓ **Paso 9:** Vemos que a la izquierda la variable \$listado, que antes no estaba inicializada, guarda ya el nombre y el precio del primer producto, si repetimos lo mismo vemos que listado se va llenando con los datos de los productos seleccionados.

Empezando. (Paso 1)



Captura de pantalla Visual Studio Code (Elaboración propia)

Pasos 2 y 3.

```

    <?php
    // This file contains the logic for the application
    // It receives data from the user and performs calculations
    // It then displays the results to the user
    // The code is written in PHP, which is a server-side scripting language
    // It uses variables to store data and control flow statements like if and for loops
    // It also includes comments to explain what each section does
    // The code is well-structured and follows best practices for readability and maintainability
    // It includes error handling and validation for user input
    // The overall purpose is to demonstrate how to build a simple web application using PHP
  
```

Captura Pantalla Visual Studio Code (Elaboración Propia)

```

    <?php
    // This file contains the logic for the application
    // It receives data from the user and performs calculations
    // It then displays the results to the user
    // The code is written in PHP, which is a server-side scripting language
    // It uses variables to store data and control flow statements like if and for loops
    // It also includes comments to explain what each section does
    // The code is well-structured and follows best practices for readability and maintainability
    // It includes error handling and validation for user input
    // The overall purpose is to demonstrate how to build a simple web application using PHP
  
```

Captura de pantalla Visual Studio Code (Elaboración propia.)

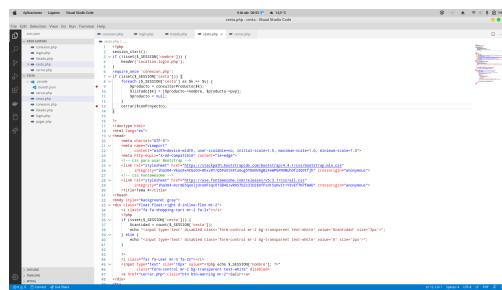
Paso 4.

```

    <?php
    // This file contains the logic for the application
    // It receives data from the user and performs calculations
    // It then displays the results to the user
    // The code is written in PHP, which is a server-side scripting language
    // It uses variables to store data and control flow statements like if and for loops
    // It also includes comments to explain what each section does
    // The code is well-structured and follows best practices for readability and maintainability
    // It includes error handling and validation for user input
    // The overall purpose is to demonstrate how to build a simple web application using PHP
  
```

Captura de pantalla Visual Studio Code (Elaboración propia)

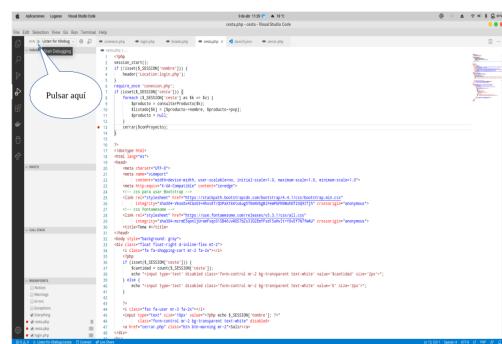
Pasos 5 y 6.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure with files like `index.php`, `lms.php`, `resource.php`, and `script.js`.
- Code Editor:** Displays PHP code for a Moodle plugin. The code includes functions for handling file uploads and displaying resource lists.
- Terminal:** Shows the command `php -S localhost:8000` running in the background.

Captura de pantalla Visual Studio Code (Elaboración propia)

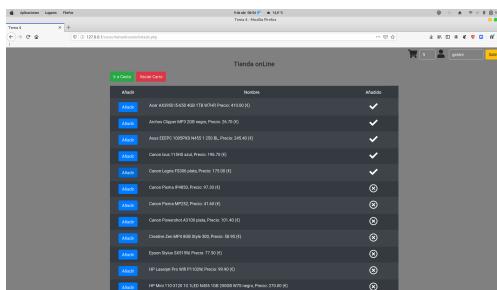


The screenshot shows the Visual Studio Code interface with a callout bubble pointing to the "Run" button in the top toolbar. The "Run" button is highlighted with a blue circle and labeled "Pulsar aquí".

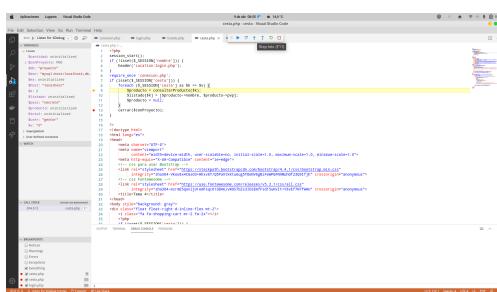
The code editor displays the same PHP code as the previous screenshot, including the `lms.php` file which contains logic for displaying course content.

Captura de pantalla Visual Studio Code (Elaboración propia)

Paso 7.

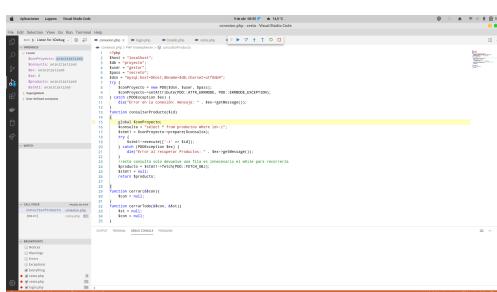


Captura de pantalla Visual Studio Code (Elaboración propia) (Elaboración propia)

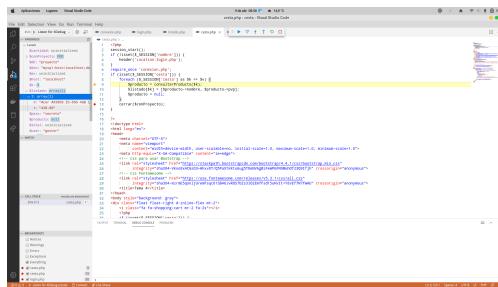


Captura de pantalla Visual Studio Code (Elaboración propia)

Pasos 8 y 9.



Captura de pantalla Visual Studio Code (Elaboración propia)



Captura de pantalla Visual Studio Code (Elaboración propia)

1 2 3 4 5 6

Recomendación

Una vez que este depurando puedes utilizar los siguientes atajos de teclas (suelen ser comunes para todos los depuradores).

- ✓ **Continuar tecla F5:** Continua la depuración.
- ✓ **Step Into tecla F1:** Ejecuta la siguiente línea de código. Si esa línea es una llamada a otra función, el programa entrará en esa función.
- ✓ **Step Over tecla F10:** Igual que la anterior, pero si la siguiente línea es una función, la ejecuta sin entrar en ella.
- ✓ **Step Out tecla Shift+F11:** Sale de la función actual.
- ✓ **Stop tecla Shift+F5:** Detiene la depuración.

Autoevaluación

Al instalar la extensión Xdebug en PHP, se instala automáticamente la extensión para Visual Studio Code, que permite gestionar la depuración del código.

- No.
 Sí.

Efectivamente, debemos instalar una extensión para poder utilizarlo como por ejemplo "PHP Debug".

Revisa los pasos que has realizado para instalar xdebug y configurar VSC para utilizarlo.

Solución

1. Opción correcta
2. Incorrecto