UNIDAD 3 – LENGUAJES DE SCRIPT DE SERVIDOR: PHP

ÍNDICE DE CONTENIDOS

1. Definición

La programación del lado del servidor permite realizar tareas que se ejecutarán en el servidor a partir de peticiones del usuario (normalmente enviadas desde el navegador) y que devolverán respuestas, generalmente en HTML.

2. Clasificación de lenguajes

Dentro de los lenguajes para desarrollar scripts de servidor tenemos:

- **CGI (Common Gateway Interface –Interfaz de Entrada Común-):** El sistema más antiguo que existe para programar páginas dinámicas de servidor. Actualmente algo desfasado por diversas razones como la dificultad con la que se desarrollan los programas, la falta de seguridad y la pesada carga que supone para el servidor. Los CGI se escriben habitualmente en el lenguaje **Perl**, pero lenguajes como C, C++ o Visual Basic pueden ser también empleados.

- **Perl**: Lenguaje interpretado, al igual que otros lenguajes de Internet como Javascript o ASP. Esto quiere decir que el código no se compila sino que cada vez que se quiere ejecutar se lee el código y se pone en marcha interpretando lo que hay escrito. Desde Perl podemos hacer llamadas a subprogramas escritos en otros lenguajes. También desde otros lenguajes podremos ejecutar código Perl.

  Si queremos trabajar con Perl será necesario tener instalado el intérprete del lenguaje. A partir de ese momento podemos ejecutar CGIs en nuestros servidores web.

- **ASP (Active Server Pages)**: Tecnología desarrollada por Microsoft. Existe una versión denominada ASP.NET, por ello las versiones anteriores a .NET se denominan ASP clásico. Con ASP se combinan HTML con los scripts y componentes COM para crear web interactivas.

- **JSP (Java Server Pages)**: Tecnología orientada a crear páginas web dinámicas con Java. Podremos crear aplicaciones web que se ejecuten en variados servidores web de múltiples plataformas, ya que Java es en esencia un lenguaje multiplataforma.

- **PHP (PHP Hypertext Preprocessor):** Lenguaje de código abierto adecuado para desarrollo web, con el fin de desarrollar scripts de servidor que puede ser incrustado dentro de páginas HTML. Es el lenguaje más extendido para desarrollar scripts de servidor dentro de las tecnologías web más habituales.

3. PHP

Dentro de todos los lenguajes de scripts de servidor que hemos mencionado en el apartado anterior elegiremos PHP como lenguaje de servidor para la asignatura.

¿Qué puede hacer PHP?

- ✓ Generar contenidos dinámicos.
- ✓ Trabajar con el sistema de ficheros del servidor (manipular ficheros y carpetas).
- ✓ Recopilar datos de formularios.
- ✓ Enviar y recibir cookies.
- ✓ Añadir, borrar, modificar datos en bases de datos.
- ✓ Controlar el acceso de usuario.

¿Por qué PHP?

- ✓ Se ejecuta en varias plataformas (Windows, Linux, Unix, Mac OS X, etc.).
- ✓ Es compatible con casi todos los servidores utilizados en la actualidad (Apache, IIS, etc.).
- ✓ Soporta una amplia gama de bases de datos.
- ✓ Es libre. Puedes descargarlo desde el recurso oficial de PHP: www.php.net.
- ✓ Es fácil de aprender y se ejecuta de manera eficiente en el lado del servidor.
- ✓ Hay multitud de librerías desarrolladas en PHP que podemos incorporar a nuestros sitios.

4. Configuración del intérprete de PHP

Intérprete es el software que ejecuta los scripts. A diferencia de los compiladores (que traducen el script a lenguaje máquina para que sea ejecutado), el intérprete ejecuta instrucción por instrucción sin traducción al lenguaje máquina.

La configuración de PHP se establece en su mayoría en el fichero **php.ini**, que contiene la asignación de valores a los diferentes parámetros. Por ejemplo:

- **short_open_tag = On** -> Activa el uso de etiquetas cortas <? ?> en vez de <?php ?>
- **file_uploads = On** -> Permite subir ficheros a nuestro servidor desde el código PHP

Para activar estas línea únicamente quitamos el punto y coma (;) que hay delante de cada parámetro.

La función en PHP **php_info();** nos muestra una página con los parámetros configurados en PHP.

5. Integración con los lenguajes de marcas

Los archivos PHP pueden contener texto, HTML, CSS, JavaScript. El código PHP se ejecuta en el servidor y el resultado se devuelve al navegador, como HTML plano normalmente, y también puede dar como resultado código Javascript, XML, CSS, PDF, archivos de imágenes, etc., es decir, lenguajes o ficheros que pueda interpretar el navegador.

Cuando el fichero ".php" es solicitado al servidor, el intérprete del mismo ejecuta las sentencias PHP transformándolas en código HTML que luego será procesado por el navegador del cliente.

PHP no está limitado a ser usado con HTML, puede ser usado junto con XHTML y cualquier tipo de ficheros XML.

6. Herramientas de edición de código

Para editar el código basta con un programa de edición de ficheros de textos planos como el bloc de notas de Windows o gedit de Ubuntu.

Para más comodidad podemos usar editores que incorporan ayudas para la edición de código como **Notepad++.**

Aunque, sin duda, lo mejor es utilizar un **IDE (Integrated Development Environment)**, software que incluye todas las herramientas para desarrollar aplicaciones. Suele incluir un editor de código con ayuda sobre sintaxis y comandos, herramientas asociadas como gestor de base de datos, etc. Destacan **Eclipse, Netbeans, Kompozer** o **Dreamweaver** (se puede descargar una versión de prueba).

7. Sintaxis del lenguaje PHP

Un script PHP se puede colocar en cualquier parte dentro del documento .php. Un script PHP comienza con <?php y termina con ?>.

Un archivo PHP normalmente contiene etiquetas HTML, y algo de código PHP.

Ejemplo que muestra el mensaje "H o l a". La orden **echo** muestra datos o textos en pantalla.

```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo </title>
</head>
<body>
<? echo "H o l a"; ?>
</body>
</html>
```

8. Comentarios en PHP

Para colocar comentarios dentro del código PHP y aclarar el código de cara a una mejor compresión cuando debamos rectificar o ampliar nuestro código podemos usar:

- // al principio de una línea
- # al principio de una línea
- Comenzar con /* y terminar el comentario con */

```
<?
// comentario
echo “abc”;
# comentario
echo “cba”;
/* comentario
comentario
comentario
*/
?>
```

9. Variables en PHP

Hay que tener en cuenta:

- Para crear una variable en PHP basta con asignarle un valor. No hay que hacer ninguna declaración como en otros lenguajes (Java, Javascript, C, C++, etc.).
- Comienzan por $, después de este símbolo debe ir una letra ó _
- Si el valor que deseamos guardar en la variable es un texto (cadena de caracteres) debe ir entre comillas.
- PHP es sensible a mayúsculas y minúsculas en los nombres de las variables.

```
<?
$a = 7;
$b = ”El número es: “;
echo $b;
echo $a;
?>
```

En este ejemplo creamos 2 variables ($a y $b). El resultado visible en la página será: El número es: 7

10. Ámbito de las variables

Se entiende por ámbito el “alcance” de las variables, es decir, dónde se pueden usar dentro del código. Hay tres ámbitos: **local**, **global** y **estático**.

Una variable **declarada dentro de una función es una variable local** y solo se puede acceder dentro de esa función. No afecta a otras en el script. En la siguiente función *echo* escribe 2 porque el $a de la función no tiene nada que ver con $a de fuera de la función.

```
<?
function funcion()
{
   $a = 1;
   echo $b;
}
$a = 2;
$b = 3;
funcion();
echo $a;
?>
```

Una variable **declarada fuera de una función es una variable global** y se puede acceder a ella desde cualquier parte del script global, pero no dentro de las funciones. En el ejemplo anterior el *echo* que hay dentro de la función no escribirá nada porque $b es una variable global (y no local). Para poder hacer referencia desde dentro de una función a una variable global es necesario escribir **global $variable**.

```
$a = 1;
$b = 2;

function Suma()
{
   global $a, $b;
   $b = $a + $b;
}

suma();
echo $b; //Muestra 3
?>
```

Las variables estáticas **se declaran dentro de una función y guardan su valor** para futuras llamadas a la función.

```
<?php
function test()
{
   static $a = 0;
   echo $a;
   $a++;
}
?>
```

Ahora, *$a* se inicializa únicamente en la primera llamada a la función, y cada vez que la función *test()* es llamada, imprimirá el valor de *$a* y lo incrementa.

11. Variables superglobales

Son variables internas que están **disponibles siempre en todos los ámbitos del script**, es decir, también desde las funciones. Algunas variables predefinidas en PHP son superglobales, lo que significa que están disponibles en todos los ámbitos a lo largo del script. Son, entre otras:

- $_SERVER, indica información sobre parámetros del servidor (nombre del servidor, puerto del usuario, etc.).
- $_GET, son los datos recibidos después de un envío desde un formulario por el método GET.
- $_POST, Igual que el anterior pero por el método POST.
- $_REQUEST, todos los datos recibidos de un formulario (por método GET o POST).
- $_SESSION, datos guardados para su uso a través de múltiples páginas.
- $_COOKIE, datos guardados por el servidor en el PC del usuario.
- $_FILES, ficheros recibidos después de un envío.

12. La orden echo

Una de las órdenes más usadas es ***echo***, sirve para mostrar datos en pantalla, en el documento HTML generado.

Diferentes formas de usar echo para mostrar la frase “Aprendiendo PHP”:

```
<?
echo “Aprendiendo PHP”;          //Escribe texto entrecomillado
echo “Aprendiendo ”, “PHP”;      //Escribe dos textos entrecomillados
echo “Aprendiendo ”. “PHP”;      //Concatena dos textos entrecomillados
$a = "PHP";
```

```
$b = "Aprendiendo ";
echo $b,$a;                 //Escribe el contenido de 2 variables
echo $b; echo $a;           //Escribe el contenido de 2 variables
echo “Aprendiendo $a”;      //Escribe cadena sustituyendo variable por valor
echo "$b$a";                //Escribe cadena sustituyendo variables por valor
?>
```

13. Tipos de datos

En PHP podemos trabajar con diferentes tipos de datos:

- Tipo **cadena**, como hemos visto anteriormente, se escriben entre comillas
- Tipo **numérico**, se escribe sin comillas y pueden ser dígitos con o sin la coma decimal.
- Tipo **booleano**, tienen dos valores true y false
- **Arrays** (vectores y matrices), colecciones de datos con un mismo nombre y un índice
- **Objetos**, datos complejos que almacenan información y formas de manipular esa información.

```
<?
$a = 1;
$b = 2.5;
$c = true;
$d = “Dato”;
$vector = array ( “rojo” , ”azul” , ”verde” );
?>
```

14. Cadenas

Una cadena es una secuencia de caracteres. El operador punto (.) concatena cadenas, por ejemplo el siguiente código produce la salida: “Este módulo profesional es: Implantación de Aplicaciones Web”.

```
<?
$a = “Implantación de “;
$b = “ Aplicaciones”;
$c = “ Web”;
$d = “Este módulo profesional es :”.$a.$b.$c;
echo $d;
?>
```

Para realizar otra seríe de operaciones tenemos funciones para procesar las cadenas:

- **strlen** nos indica la longitud, número de caracteres, de la cadena
- **strtolower** convierte la cadena a minúsculas
- **strtoupper** convierte la cadena a mayúsculas
- **substr** devuelve un trozo de una cadena
- **strpos** busca una cadena dentro de otra y devuelve la posición
- **trim** elimina espacios en blanco

```
<?
echo strlen(“hola”);          //devuelve 4
echo strtoupper(“hola”);      //devuelve “HOLA”
echo strpos(“hola”,”o”);      //devuelve 1, la primera posición es 0.
echo substr(“hola”,1,3);      //devuelve “ola”
echo trim(“ hola “);          //devuelve “hola”
?>
```

15. Operadores

+	Suma	Sumar
-	Resta	Restar
*	Multiplicación	Multiplicar
/	División	Dividir
%	Resto	Resto de una división
++	Incremento	Sumar 1 al valor de una variable, finalidad contar iteración o similares
--	Decremento	Restar 1 al valor de una variable
==	Comparación de igualdad	Comparar si dos expresiones son iguales
!=	Comparación de desigualdad	Comparar si dos expresiones son diferentes
<	Comparación menor	Comparar si una expresión es menor que otra
>	Comparación mayor	Comparar si una expresión es mayor que otra
!	Operador de negación	Negar una expresión booleana
&&	Operador Y	Combinar dos expresiones booleanas para ver si las dos se cumplen
||	Operador O	Combinar dos expresiones booleanas para ver si una de las dos se cumple

16. Estructuras de control

16.1. Bifurcaciones condicionales

16.1.1. if

```
if (condicion)
{ secuencia de comandos si se cumple la condición }
else if     //Opcional. Tantos como "else if" como sea necesario
{ secuencia de comandos si NO se cumple la condición }
else        //Opcional. Como máximo, un "else"
{ secuencia de comandos si NO se cumple la condición }
```

16.1.2. switch

```
switch(variable)
{
case valor1:
secuencia de comandos si variable vale valor1
break;
case valor2:
secuencia de comandos si variable vale valor2
break;
...
default:
secuencia de comandos si variable NO vale ninguno de los anteriores
break;
}
```

Ejemplo: Si un alumno ha aprobado se emite el título, si no, no se hace nada.

```
if ($aprobado=="SI"
{ emitirtitulo(); }
```

Ejemplo: Dependiendo de si es familia numerosa tipo 1, 2 se aplica un 5% o un 10% de descuento, si no, no se aplica ningún descuento:

```
switch($tipofamilianumerosa)
{
case 2:
$descuento=0.10;
```

```
break;
case 1:
$descuento=0.05;
break;
default:
$descuento=0;
break;
}
```

16.2. Bucles

PHP dispone de diferentes tipos bucles o iteraciones:

16.2.1. while

Si no se cumple la condición, las instrucciones no se ejecutan ni una vez.

```
while (condición)
{ instrucciones que se repiten mientras se cumpla la condición }
```

16.2.2. do … while

Diferencia con while: Como mínimo, las instrucciones se ejecutan una vez.

```
do
{ instrucciones que se repiten mientras se cumpla la condición }
while (condición)
```

16.2.3. for

```
for(inicio; condición; incremento*)
{ instrucciones que se repiten desde que la situación expresada en inicio hasta el final que expresa la condición (mientras ésta se cumpla) realizando un el incremento en cada iteración }
```

***NOTA**: También podría ser un decremento (sería un incremento negativo).

16.2.4. foreach

```
foreach (elemento en colección)
{ secuencia de comandos que se repiten por cada uno de los elementos que hay en la colección}
```

17. Funciones (nativas y de usuario)

En PHP hay una gran cantidad de funciones como las que vimos para manipular cadenas. Una referencia completa de estas se puede encontrar en: http://php.net/manual/es/funcref.php

Además podemos encontrar infinidad de librerías desarrolladas para realizar diferentes tareas, por ejemplo, en http://www.fpdf.org/ hay un librería con ejemplos detallados para generar archivos PDF desde PHP.

Y, por supuesto, el programador puede diseñar sus propias funciones para tareas a las que no encuentre, ni en la referencia del lenguaje, ni en recursos de internet, una solución.

Para diseñar una función se utiliza la siguiente sintaxis:

```
function nombredelafuncion()
{ código a ejecutar }
```

Para hacer uso de la función solo es necesario **invocarla** desde cualquier lugar con nombredefuncion(); al diseñar una función también podemos enviarle datos para que los procese dentro de su código, estos son los denominados parámetros:

```
function nombredelafuncion(parámetro1, parámetro2,…)
{ código a ejecutar }
```

Ejemplo, función que muestra en negrita los datos enviados en pantalla:

```
<?
function negrita($dato)
{
 echo “<strong>”;
echo $dato;
echo “</strong>”;
}
echo “Nombre:”;
negrita(“José”);
echo “ Apellidos:”;
negrita(“García García”);
?>
```

Las funciones pueden devolver datos, para ello emplearos la instrucción **return**. El dato devuelto puede pertenecer a cualquier de los vistos anteriormente (numéricos, booleanos, cadenas, arrays, etc.).

18. Gestión de errores

En PHP los errores más comunes son errores de sintaxis (alguna instrucción o parte del código está mal escrita). Cuando se produce un error se envía información sobre el mismo (nombre de archivo, línea y descripción del error) al navegador.

Al ejecutar el script nos encontraremos errores típicos y sus motivos como estos:

```
Parse error:  syntax error, unexpected …., expecting ',' or ';' in archivo.php on line xx
```

(Se ha olvidado un punto y coma al final de la instrucción que está en la línea xx)

```
Parse error: syntax error, unexpected 'símbolo' in archive.php on line xx
```

(Suele producirlo paréntesis o llaves que no se han abierto y se han cerrado en la línea xx)

```
Parse error: syntax error, unexpected end of file in arhivo.php on line xx
```

(Se ha olvidado cerrar un paréntesis o una llave)

```
Fatal error: Call to undefined function xxxx() in arhivo.php on line xx
```

(El nombre de la función xxxx es incorrecto)

```
Notice: Undefined variable: xxxx in archivo.php on line xx
```

(El nombre de una variable es incorrecto, recordar que hay que poner $ al principio)

Aún cuando una sentencia o expresión sea sintácticamente correcta, puede causar un error al intentar ejecutarla. Los errores que se detectan en la ejecución se llaman **excepciones**.

En PHP, como en muchos lenguajes (Java), se pueden gestionar estas excepciones, es decir, evitar una terminación anormal del script y hacer una salida que evite el mensaje de error para que el usuario no se quede sin opciones a seguir trabajando con la aplicación web.

El tratamiento más sencillo para evitar estas excepciones es el uso de la función **die()**, que detiene el programa en el punto en que la situamos y emite, si deseamos, un mensaje.

Ejemplo para evitar un mensaje de error de división por cero:

```
<?
$divisor=0; //Probar con cero y otro número para ver la diferencia
$dividendo=10;
if ($divisor==0)
{ die("Error al intentar dividir por cero"); }
else
{ echo "Resultado:", $dividendo/$divisor;}
?>
```

Otra forma, más elegante, de tratar estas excepciones es con **try catch finally**, que consiste en intentar (try) un bloque de código, si se produce una excepción la tratamos con otro bloque (**catch**) y finalmente se ejecuta otro bloque de código (**finally**). Para que se ejecuten estos códigos hay que lanzar la excepción con **throw**.

19. Formularios

Un script tiene sentido cuando a partir de unos datos realiza unas tareas. Para recoger estos tenemos varias formas, la más habitual es el uso de formularios. Estos irán incorporados dentro del código HTML. Las etiquetas clásicas (funcionan en cualquier navegador) son las de la tabla. En HTML5 se han incorporado muchos controles nuevos para capturar datos específicos como e-mail, fecha, rangos, etc.

<form action=" " method=" "></form>	Los datos recogidos dentro de esta etiqueta se enviarán a la página que indiquemos en **action** usando el método indicado en **method**.
<select name="name"></select>	Lista desplegable para captura datos que serán identificados con name.
<option value="enviado">Visible </option>	Para cada dato (ítem) de la lista desplegable en **value** se indica el valor a enviar y dentro de las etiquetas el valor que visualiza el usuario.
<textarea name="..."></textarea>	Para recoger comentarios, observaciones y similares.
<input type="checkbox" name="name">	Casilla de verificación.
<input type="radio" name=".." value=" ">	Botones de opción, el nombre debe coincidir dentro del grupo, el valor que se envía es el **value** de la opción marcada idenficada con el nombre común a todas las opciones
<input type="submit" value="name">	Botón que al pulsar envía los datos capturados en el formulario a la página indicada en action de la etiqueta **form**.
<input type="reset">	Borra los datos tecleados en el formulario por el usuario.

Hay dos métodos de envío de datos desde un formulario:

- GET, los datos enviados aparecen en la URL destino.
- POST, los datos no son visibles en la URL.

Ejemplo

```
<form action="procesar.php" method="post">
<input type="text" name="dato">
<input type="submit">
</form>
```

NOTA: También podemos enviar datos utilizando técnicas como AJAX, que hacen una comunicación interactiva con el servidor, es decir, desde la propia página HTML enviamos y recibimos datos, actualizando a partir de los datos recibidos el código HTML de la página.

19.1. Procesado de la información recibida de un formulario

Toda esta información enviada desde los formularios es recibida por los scripts de PHP mediante las variables superglobales $_GET (si el método de envío del formulario fue GET), $_POST (enviados por POST), $_REQUEST (enviadas por cualquier método), $_FILES (ficheros subidos desde los formularios).

Ejemplo de procesado del dato recogido en el apartado anterior

```
<?
echo “El dato recibido es: ”;
echo $_POST[“dato”];
?>
```

20. Envío de ficheros

Para subir ficheros es necesario que el formulario de recogida de datos cumpla:

- Método de envío **post**
- La etiqueta formulario debe tener el atributo **enctype="multipart/form-data"**
- Para seleccionar el fichero hay que usar la etiqueta **<input type=”file”>**

```
<form action="subida.php" method="post" enctype="multipart/form-data">
Seleccionar imagen:
<input type="file" name="fichero" >
<input type="submit">
</form>
```

El fichero se recibe mediante la variable $_FILE, va a una **carpeta temporal**.

Por seguridad, es conveniente realizar una serie de chequeos en el fichero recibido:

- Comprobar el tamaño con **$_FILES["fichero"]["size"]**
- Comprobar el tipo de fichero con **$_FILES["fichero"]["type"]**
- Comprobar si ya existe un fichero con el mismo nombre con **file_exists()**
- …

Una vez que todo este correcto, movemos el fichero a su carpeta final:

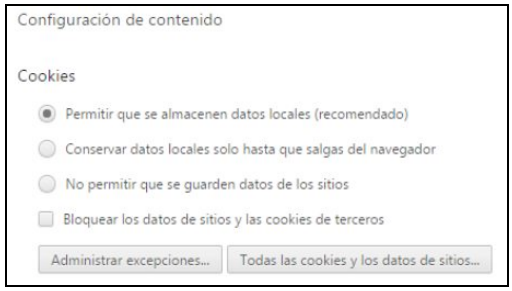
```
move_uploaded_file($_FILES["fichero"]["tmp_name"],destinofinal)
```

21. Cookies

Las cookies son informaciones que se guardan en el PC del usuario que visita la web. Son guardadas en ficheros de texto en el PC del usuario y tienen una fecha de caducidad.

Desde la mayoría de los navegadores podemos ver las cookies que tenemos almacenadas en nuestro ordenador. Las cookies cambian de un PC a otro PC; dentro del mismo PC, de un usuario a otro; dentro del mismo PC y con el mismo usuario, de un navegador a otro.

La siguiente pantalla muestra el acceso a las cookies almacenadas en Chrome (Configuración avanzada – configuración de contenido):



Desde PHP se pueden guardar y recuperar estas cookies del ordenador del usuario:

- Para leer una cookie utilizamos la variable superglobal **$_COOKIE[“nombre”]**
- Para guardar se usa la función **setcookie**

Ojo, es incorrecto ~~$_COOKIE[“nombre”]=valor;~~

Ejemplo. Guardar en una cookie el nombre del usuario, caduca al año (365 días * 24 horas * 60 minutos * 60 segundos son los segundos de un año, se los sumamos a la fecha actual):

```
setcookie(“usuario”,”Juan”,time()+60*60*24*365);
```

Cuando queramos consultar el nombre del usuario basta con:

```
echo $_COOKIE[“usuario”];
```

Para borrar una cookie basta poner una fecha de caducidad time()+0.

Cuidado, si en una página guardamos una cookie no estará disponible hasta la próxima visita.

22. Sesiones

Una sesión es una comunicación mantenida entre el cliente (navegador normalmente) y el servidor. En PHP podemos usar variables de sesión ($_SESSION) para mantener información mientras que el usuario navega por múltiples páginas.

La sesión tiene una fecha de caducidad que se configura en el fichero **php.ini** (parámetro *session.cookie_lifetime*), aunque también podemos destruir con la función **session_destroy();**

Todas las páginas que visite el usuario deben tener la función **session_start()** al comienzo del código PHP, si no, se destruye la sesión.

Mientras el usuario mantenga abierto el navegador y siga interactuando con el servidor la sesión se mantendrá abierta. Cuando el usuario cierra el navegador o no interactúa en el tiempo estipulado en el parámetro indicado anteriormente se acaba la sesión.

Ejemplo, primera página web:

```
<?
session_start();
$_SESSION[“paginas_visitadas”]=1;
echo “Ha visitado: “, $_SESSION[“paginas_visitadas”],” páginas”;
?>
<a href=”pagina2.php”>Visitar página 2</a>
```

La página anterior mostraría “Ha visitado 1 páginas”. Visitar página 2

La segunda página web sería:

```
<?
session_start();
$_SESSION[“paginas_visitadas”]++;
echo “Ha visitado: “, $_SESSION[“paginas_visitadas”],” páginas”;
?>
<a href=”pagina3.php”>Visitar página 3</a>
```

La página anterior mostraría Ha visitado 2 páginas. Visitar página 3

Si el usuario sigue moviéndose entre páginas el contador de visitas sigue subiendo.

23. Autentificación de usuarios y control de accesos

Por seguridad, es habitual registrar los accesos a una web, por ejemplo la IP pública que está accediendo. Normalmente estos datos se registran en una BD para consultar quién accede, llevar estadísticas de visitas, etc.

Para la autentificación de usuario se suele usar usuario y contraseña, que son recogidos mediante un formulario y enviados a una página que verifica si son correctos, para ello lo habitual es cotejarlos con BD. Por seguridad, los datos en la BD suelen estar cifrados. PHP ofrece funciones muy útiles para manipular datos "delicados", funciones de encriptación, etc. **El uso de bases de datos es cuestión de la próxima unidad**.