Desarrollo Web en entorno servidor

Unidad 3:
Aplicaciones Web en PHP
Manejo de ficheros y formato de datos

Tabla de contenido

1. Lectura y escritura de ficheros. 2
   1.1. Especificadores de conversión 4
   1.2. Funciones para manejar cadenas con formatos 5
   1.3. Funciones para leer y escribir cadenas con formatos en ficheros 7
2. Registro de errores 9
3. Otras funciones para trabajar con ficheros 10

1. Lectura y escritura de ficheros.

En PHP hay varias librerías para manejo de ficheros. La más importante es Filesystem, que incluye funciones para leer y escribir ficheros y para manejar el sistema de archivos Son funciones muy similares a las de manejo de ficheros en C.

Para abrir un fichero se usa la función `fopen()`, indicando la ruta del fichero y el modo en el que se abre. Si hay algún error, la función devuelve FALSE; en otro caso devuelve un puntero para manejar el fichero.

Los modos de apertura de los ficheros son:

Modo	Descripción
r	Solo lectura. Si el fichero no existe, devuelve FALSE
r+	Lectura y escritura. Si el fichero no existe, devuelve FALSE
w	Solo escritura. Si el fichero no existe, se crea; si existe, se trunca, es decir, se borra el contenido anterior. Si no puede crearlo, devuelve FALSE
w+	Lectura y escritura. Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo devuelve FALSE
a	Solo escritura. Las escrituras se realizan siempre al final de fichero. Si el fichero no existe, se crea. Si no puede crearlo, devuelve FALSE
a+	Lectura y escritura. Las escrituras se realizan siempre al final de fichero. Si el fichero no existe, se crea. Si no puede crearlo, devuelve FALSE
x	Escribir solamente. Crea un nuevo archivo. Devuelve FALSE y un error si el archivo ya existe
x+	Lectura y escritura. Crea un nuevo archivo. Devuelve FALSE y un error si el archivo ya existe
c	Escribir solamente. Abre el archivo; o crea un nuevo archivo si no existe. Coloca el puntero de archivo al principio del archivo
c+	Lectura y escritura. Abre el archivo; o crea un nuevo archivo si no existe. Coloca el puntero de archivo al principio del archivo

En el ejemplo `ficheros_fopen.php` vemos los casos de poder abrir un fichero o no poder abrirlo.

```
1:  <?php
2:      $fich = fopen("fichero_ejemplo.txt", "r");
3:      if ($fich === False){
4:          echo "No se encuentra fichero_ejemplo.txt<br>";
5:      }else{
6:          echo "fichero_ejemplo.txt se abrió con éxito<br>";
7:      }
8:      $fich = fopen("fichero_no_existe.txt", "r");
```

```
 9:      if ($fich === False){
10:          echo "No se encuentra fichero_no_existe.txt<br>";
11:      }else{
12:          echo " fichero_no_existe.txt se abrió con éxito<br>";
13:      }
```

El primer fichero sí existe, pero el segundo no. Al ejecutarlo la salida será:

```
fichero_ejemplo.txt se abrió con éxito

Warning: fopen(fichero_no_existe.txt): Failed to open stream: No such file or directory
in C:\xampp\htdocs\DWES\_Pruebas\ficheros_fopen.php on line 8

No se encuentra fichero_no_existe.txt
```

La lectura y la escritura se realizan a partir del indicador de posición del fichero. Cuando se abre un fichero, el indicador se sitúa al inicio de este, salvo si se ha abierto en modo "a" o "a+" que se sitúa al final del mismo. Cuando se lee o escribe, el puntero avanza para situarse al final del último carácter o byte leído o escrito.

En ejemplo `ficheros_fgetc.php`, se abre un fichero en modo lectura y se lee carácter a carácter usan do la función `fgetc()`, que devuelve el siguiente carácter a partir del indicador de posición.

```
 1:  <?php
 2:      $fich = fopen("fichero_ejemplo.txt", "r");
 3:      if ($fich === False){
 4:          echo "No se encuentra el fichero o no se pudo leer<br>";
 5:      }else{
 6:          while( !feof($fich) ){
 7:              $car = fgetc($fich);
 8:              echo $car;
 9:          }
10:      }
11:      fclose($fich);
```

Para detectar el final del fichero se usa la función `feof()`, que devuelve TRUE cuando ya se ha llegado al final. Cuando se termina de trabajar con el fichero, se cierra con `fclose()`.

Los ficheros no pueden estar abiertos por el usuario para ver su contenido si se están manipulando mediante código, por lo que siempre se ha de tener la precaución de tenerlos convenientemente cerrados.

La utilidad del manejo de los ficheros radica en poder trabajar con información con un determinado formato. Por ejemplo, el fichero matriz.txt representa una matriz de 4 filas y 4 columnas.

```
23 234 611 5
1233 565 123 5
123 54 757 12
77 88 9 99
```

Para indicar que cada línea está formada por 4 números separados por espacios se usa como formato: "%d %d %d %d", donde cada %d es un especificador de conversión.

1.1. Especificadores de conversión

Cada especificación de conversión consiste en un signo de porcentaje (%), seguido por uno o más de estos elementos, en orden:

1. Un *especificador de signo* opcional que fuerza a usar un signo (- o +) en un número.
2. Un *especificador de relleno* opcional que indica qué carácter se utiliza para rellenar el resultado hasta el tamaño justo del string. Este puede ser un carácter de espacio o un 0. El valor por defecto es rellenar con espacios. Un carácter de relleno alternativo se puede especificar prefijándolo con una comilla simple (').
3. Un *especificador de alineación* opcional que indica si el resultado debe ser alineado a la izquierda o a la derecha. El valor por defecto es justificado a la derecha, un carácter - lo justificará a la izquierda.
4. Un número opcional, un *especificador de ancho* que indica de cuántos caracteres (mínimo) resultará esta conversión.
5. Un *especificador de precisión* opcional en la forma de un punto (.) seguido de un string opcional de dígitos decimales que indica cuántos dígitos decimales deben mostrarse para los números de punto flotante. Cuando se utiliza este especificador con un string, actúa como un punto de corte, estableciendo un límite máximo de caracteres al string. Además, el carácter empleado cuando se rellena un número podría especificarse opcionalmente entre el punto y el dígito.
6. Un *especificador de tipo* que indica con qué tipo deben ser tratados los datos del argumento. Los tipos posibles son:

    - % - un carácter de porcentaje literal. No se requiere argumento.
    - b - el argumento es tratado como un valor de tipo integer y presentado como un número binario.
    - c - el argumento es tratado como un valor de tipo integer y presentado como el carácter con ese valor ASCII.
    - d - el argumento es tratado como un valor de tipo integer y presentado como un número decimal (con signo).
    - e - el argumento es tratado con notación científica (e.g. 1.2e+2).
    - E - como %e pero utiliza la letra mayúscula (e.g. 1.2E+2).
    - f - el argumento es tratado como un valor de tipo float y presentado como un número de punto flotante (considerando la configuración regional).
    - F - el argumento es tratado como un valor de tipo float y presentado como un número de punto flotante (no considerando la configuración regional).
    - g - lo mismo que %e y %f.
    - G - lo mismo que %E y %f.
    - o - el argumento es tratado como un valor de tipo integer y presentado como un número octal.
    - s - el argumento es tratado y presentado como un string.
    - u - el argumento es tratado como un valor de tipo integer y presentado como un número decimal sin signo.
    - x - el argumento es tratado como un valor de tipo integer y presentado como un número hexadecimal (con las letras en minúsculas).
    - X - el argumento es tratado como un valor de tipo integer y presentado como un número hexadecimal (con las letras en mayúsculas).

Los especificador de tipo por tipo de dato son:

Tipo	Especificador
string	s
integer	d, u, c, o, x, X, b
double	g, G, e, E, f, F

1.2. Funciones para manejar cadenas con formatos

Para poder mostrar cadenas con formato se usan las funciones printf() o sprintf(). La diferencia entre ambas es que sprintf() devuelve la cadena con los datos formateados, mientras que printf() imprime la cadena directamente. Veamos algunos ejemplos:

```
<?php
$num = 5;
$ubicación = 'árbol';
$formato = 'Hay %d monos en el %s';
echo sprintf($formato, $num, $ubicación);
```

Esto producirá "Hay 5 monos en el árbol".

Es habitual que las cadenas se formen en ficheros aparte de nuestro script principal, generalmente en los casos que trabajemos con múltiples idiomas. Esto puede dar lugar a que cuando la oración esté traducida, la posición de los parámetros cambie. También puede ser que queramos utilizarlos en una oración similar en nuestro idioma, pero con los parámetros en otro orden.

Por ejemplo, si queremos escribir la oración “El árbol contiene 5 monos” tendríamos que cambiar la última línea del ejemplo anterior, para indicar adecuadamente la posición.

Sin embargo, podemos modificar la cadena de la línea cuatro para que los argumentos se intercambien. Veamos el siguiente ejemplo:

```
<?php
$num = 5;
$ubicación = 'árbol';
$formato = 'El %2$s contiene %1$d monos';
echo sprintf($formato, $num, $ubicación);
```

Un beneficio adicional es que se pueden repetir los especificadores de conversión sin agregar más argumentos al código. Por ejemplo:

```
<?php
$num = 5;
$ubicación = 'árbol';
$formato = 'El %2$s contiene %1$d monos. Es un bonito %2$s con %1$d monos.';
echo sprintf($formato, $num, $ubicación);
```

Cuando se utiliza el intercambio de argumentos, el *especificador de posición* debe ir inmediatamente después del signo de porcentaje (%).

Veamos un ejemplo sobre los caracteres de relleno.

```php
<?php
echo sprintf("%'.9d\n", 123);
echo "<br>";
echo sprintf("%'.09d\n", 123);
```

El resultado del ejemplo sería:

```
......123
000000123
```

A continuación se muestra un ejemplo con todos los especificadores de tipo para ver sus diferentes salidas.

```php
<?php
$n =  43951789;
$u = -43951789;
$c = 65; // ASCII 65 es 'A'
printf("%%b = '%b'<br>", $n); // representación binaria
printf("%%c = '%c'<br>", $c); // muestra el carácter ascii, igual que chr()
printf("%%d = '%d'<br>", $n); // representación estándar de un entero
printf("%%e = '%e'<br>", $n); // notación científica
printf("%%u = '%u'<br>", $n); // representación sin signo de un entero positivo
printf("%%u = '%u'<br>", $u); // representación sin signo de un entero negativo
printf("%%f = '%f'<br>", $n); // representación de punto flotante
printf("%%o = '%o'<br>", $n); // representación octal
printf("%%s = '%s'<br>", $n); // representación en una cadena
printf("%%x = '%x'<br>", $n); // representación hexadecimal (minúsculas)
printf("%%X = '%X'<br>", $n); // representación hexadecimal (mayúsculas)
printf("%%+d = '%+d'<br>", $n); // especificador de signo sobre un entero positivo
printf("%%+d = '%+d'<br>", $u); // especificador de signo sobre un entero negativo
```

El resultado del ejemplo es:

```
%b = '10100111101010011010101101'
%c = 'A'
%d = '43951789'
%e = '4.395179e+7'
%u = '43951789'
%u = '18446744073665599827'
%f = '43951789.000000'
%o = '247523255'
%s = '43951789'
%x = '29ea6ad'
%X = '29EA6AD'
%+d = '+43951789'
%+d = '-43951789'
```

Por último, mostramos un ejemplo con los especificadores de cadena.

```php
<?php
$s = 'mono';
$t = 'muchos monos';
printf("[%s]<br>",      $s); // salida estándar de string
printf("[%10s]<br>",    $s); // justificación a la derecha con espacios
printf("[%-10s]<br>",   $s); // justificación a la izquierda con espacios
printf("[%010s]<br>",   $s); // rellenado con ceros también funciona con strings
```

```
printf("[%'#10s]<br>",  $s); // utiliza el carácter de relleno personalizado '#'
printf("[%10.10s]<br>", $t); // justificación a la izquierda pero con un corte a los
10 caracteres
```

El resultado del ejemplo es:

```
[mono]
[      mono]
[mono      ]
[000000mono]
[######mono]
[muchos mon]
```

Sin embargo, la salida por HTML es:

```
[mono]
[ mono]
[mono ]
[000000mono]
[######mono]
[muchos mon]
```

La diferencia radica en que en HTML los espacios en blanco añadidos en las sentencias dos y 3 colapsan en uno solo. Podemos hacer uso de las etiquetas <pre> y </pre> para que se muestre la salida con el formato originalmente generado por php.

1.3. Funciones para leer y escribir cadenas con formatos en ficheros

Para la lectura y escritura de los ficheros con un formato determinado se utilizan las funciones `fscanf()` y `fprintf()`, respectivamente.

`fscanf()` lee un fichero hasta el primer salto de línea y le aplica un formato.

Hay dos maneras de usarla. En la primera, se le pasan dos parámetros y devuelve un array con los valores leídos.

```
$valores = fscanf($fichero, $formato);
```

También se le pueden pasar variables adicionales para que almacene en ellas los valores leídos en lugar de devolverlos. En este caso devuelve el número de valores leídos correcta mente.

```
$valores = fscanf($fichero, $formato, $var1, ...)
```

El ejemplo `ficheros_fscanf.php` muestra cómo recorrer un fichero con el formato anterior utilizando `fscanf()` de las dos maneras posibles. Antes de empezar con la segunda vuelta, sitúa el indicada de posición de nuevo al principio del fichero usando `rewind ()`.

```
 1:  <?php
 2:     $fich = fopen("matriz.txt", "r");
 3:     if ($fich === False){
 4:         echo "No se encuentra el fichero o no se pudo leer<br>";
 5:     }else{
 6:         while( !feof($fich) ){
 7:             $num = fscanf($fich, "%d %d %d %d");
 8:             echo "$num[0] $num[1] $num[2] $num[3] <br>";
 9:         }
10:     }
```

```
11:     rewind($fich);
12:     while( !feof($fich) ){
13:             $num = fscanf($fich, "%d %d %d %d", $num1, $num2, $num3, $num4 );
14:             echo "$num1 $num2 $num3 $num4 <br>";
15:         }
16:     fclose($fich);
```

Por otra parte, la función `fprintf()` escribe en un fichero una cadena de caracteres con el formato indicado, sustituyendo cada especificador de conversión por la correspondiente variable. Un ejemplo de uso está en `ficheros_fprintf.php`

```
1:  <?php
2:      $fich = fopen('fecha.txt', 'w');
3:      if ($fich === False){
4:          echo "Error al abrir o crear el fichero<br>";
5:      } else{
6:          fprintf($fich, "%04d-%02d-%02d", "1745", "1", "3");
7:      }
```

Esta función no inserta ningún salto de línea, por lo que el puntero, una vez ejecutada esta función, se encuentra al final de la última línea. Podemos añadir un salto de línea añadiendo la cadena de caracteres '\n'

También son útiles las funciones `file_get_contents()` y `file_put_contents()`. La primera devuelve una cadena con el contenido de un fichero. La segunda hace lo contrario, escribe datos en un fichero, si no existe, lo crea. La ventaja que tienen es que funcionan directamente a partir de la ruta fichero, así que no hace falta llamar a `fopen()` y `fclose()`.

Lo podemos ver en el ejemplo, ficheros `ficheros_file_get_contents.php`

```
1:  <?php
2:      $contenido = file_get_contents("fichero_ejemplo.txt");
3:      echo "Contenido del fichero: $contenido<br>";
4:      $res = file_put_contents("fichero_salida.txt", "Fichero creado con
file_put_contents");
5:      if($res){
6:          echo "Fichero creado con éxito";
7:      }else{
8:          echo "Error al crear el fichero";
9:      }
```

Para que la función `file_put_contents()` no sobrescriba el archivo con el contenido nuevo se ha de añadir un tercer argumento en su llamada, que puede tomar tres valores:

FILE_USE_INCLUDE_PATH: Busca el archivo en el directorio incluido.

FILE_APPEND: Si el fichero ya existe, añade la información al fichero en vez de sobrescribirlo.

LOCK_EX: Adquirir acceso exclusivo al fichero mientras se está ejecutando la escritura.

La función `file_get_contents` también admite otros parámetros, que permiten especificar la ruta del archivo o determinar el número de caracteres que se desean extraer.

2. Registro de errores

Uno de los principales usos en la lectura y escritura de ficheros es la creación de registros de errores o eventos, conocidos habitualmente como logs de errores o de eventos, en las aplicaciones web. Su creación es muy sencilla en PHP.

Vamos a recuperar el ejemplo `error_handler.php` y lo modificamos para crear dichos registros de errores.

Veamos `error_handler_log.php`

```
 1:  <?php
 2:     function manejadorErrores($errno, $str, $file, $line){
 3:         /* Originalmente mostrabamos toda esta información, ahora la guardamos
en un registro.
 4:         echo "Ocurrió el error: $errno <br>";
 5:         echo "Nombre del error: $str <br>";
 6:         echo "Archivo del error: $file <br>";
 7:         echo "Línea del error: $line <br>";
 8:         */
 9:         $log = fopen('error.log','a');
10:         fwrite($log, "[".date("r")."]. Error $errno, línea $line, archivo $file:
$str\n");
11:         fclose($log);
12:     }
13:     set_error_handler("manejadorErrores");
14:     $a=3;
15:     $a = $b; // causa error, $b no está inicializada
```

En la línea 10 utilizamos la función `fwrite()`, la cual simplemente escribe una cadena en un fichero. También usamos la función `date()` con el parámetro '`r`'. Para mas formatos de fechas, consultar PHP: date - Manual.

También podemos llamar a la función `manejadorErrores()` para crear nuestro registro propio de errores. Lo vemos en el ejemplo `comprobar_cookie.php`.

```
1:  <?php
2:  include "error_handler_log.php";
3:  // Si no existe la cookie sesion
4:  if(!isset($_COOKIE['sesion'])){
5:      // Guardamos un error
6:      manejadorErrores('001','No existe la cookie de sesión',
$_SERVER['PHP_SELF'],4);
7:  }
```

Para poder hacer uso de la función `manejadorErrores()`incluimos el script correspondiente en la línea 2.

3. Otras funciones para trabajar con ficheros

Función	Descripción
`fgets($fich, $pos)`	Devuelve una cadena con los caracteres desde el indicador de posición
`fputs($fich, $cad)`	Escribe una cadena en el fichero
`fseck($fich, $pos)`	Sitúa el cursor del fichero en la posición indicada
`ftell($fich)`	Devuelve el indicador de posición del fichero
`rewind($fich)`	Sitúa el indicador de posición al principio del fichero
`fopen($ruta, modo)`	Abre un fichero
`fclose($fich)`	Cierra un fichero
`fread($fich, $longitud)`	Lee $longitud de bytes
`fwrite($fich, $cadena)`	Escritura de una cadena en un fichero
`copy($origen, $destino)`	Copia un fichero
`unlink($borra)`	Borra un fichero
`move($actual, $nuevo)`	Mueve un fichero
`filesize($ruta)`	Devuelve el tamaño del fichero en bytes
`filetype($ruta)`	Devuelve el tipo de fichero
`is_file($ruta)` `is_dir $ruta)`	Para comprobar si la ruta corresponde a un fichero Para comprobar si la ruta corresponde a un directorio
`rename($actual, $nuevo)`	Cambia el nombre a un fichero

Ejercicio 2

Crear un formulario de registro. Este formulario de registro ha de tener dos campos: usuario y clave. Una vez enviados los datos del formulario, estos se han de guardar en el fichero “acceso.txt”.

Cada una de las líneas del fichero “acceso.txt” se compone de un nombre de usuario, una clave y un rol, que por defecto será 0. Todos ellos separados por un espacio en blanco es decir:

Usu1 Cla1 0

Usu2 Cla2 0

El usuario no puede estar vacío y solo puede contener caracteres alfanuméricos sin espacios. En caso contrario mostramos un mensaje de advertencia en el formulario.

La clave no puede estar vacía, contener espacios en blanco y solo contener caracteres alfanuméricos. En caso contrario mostramos un mensaje de advertencia en el formulario.

Si el usuario introducido para registrarse ya existe en el fichero, entonces no guardamos los datos y mostramos un mensaje de advertencia en el formulario.

Ejercicio 3

Modificar la función `comprobar_ususario($nombre, $clave)` del ejemplo `sesiones1_login.php` para que valide los datos introducidos según los datos almacenados en el fichero “acceso.txt” comprobando usuario y contraseña.