Uso de funciones

Al igual que en otros lenguajes de programación (como por ejemplo JavaScript), las funciones en PHP nos van a permitir encapsular un conjunto de instrucciones bajo un nombre, y poderlo ejecutar en bloque cada vez que lo necesitemos, simplemente utilizando el nombre que le hayamos puesto a la función.

1. Definición de funciones

Las funciones en PHP se definen con la palabra `function` (como en JavaScript), seguida del nombre de la función y unos paréntesis. A diferencia de otros lenguajes, en PHP no se distinguen mayúsculas y minúsculas en los nombres de funciones (sí se distinguen en los nombres de variables). Veamos un ejemplo:

```
function saluda()
{
    echo "Hola, buenas";
}
```

En este caso, la función muestra un mensaje de saludo. Podríamos usarla en cualquier lugar de nuestra página PHP con:

```
saluda();
```

y mostraría en la página web el texto "Hola, buenas" en el lugar donde la hubiéramos colocado.

Notar que estos dos fragmentos de código pueden ir en bloques PHP diferentes. Por ejemplo, podemos definir las funciones al inicio del código PHP, y luego llamarlas después:

```
<?php
function saluda()
{
    echo "Hola, buenas";
}
?>
<!DOCTYPE html>
...
<?php
saluda();
?>
```

2. Uso de parámetros

Como en otros lenguajes, las funciones PHP también admiten unos datos entre los paréntesis, llamados *parámetros*, que sirven para proporcionar datos a la función desde fuera, al llamarla. Por ejemplo, esta función suma los dos datos (numéricos) que se le pasan como parámetros:

```
function suma($a, $b)
{
    echo $a + $b;
}
```

Y para utilizarla, simplemente la llamamos pasándole dos datos numéricos:

```
suma(3, 2.5);
```

> **Ejercicio 1:**
>
> Crea una carpeta llamada **ejercicios2** en tu carpeta de documentos de Apache, y dentro guarda los ejercicios de esta sesión. Crea una página llamada **saludo_funciones.php**. Crea en ella una función llamada `saludo($n)`, y pon dentro el código para que saque un saludo con el nombre que se le pase como parámetro. Por ejemplo, "Hola, Nacho". Después, llama a esta función desde el código PHP para que muestre un saludo en la página con el nombre que quieras.

> **Ejercicio 2:**
>
> Crea una página llamada **contador_funciones.php** en la carpeta de ejercicios de la sesión. Crea una función llamada `cuenta($a, $b)` que reciba dos parámetros y vaya contando de un número al otro, separando los números por comas. Después, pruébala en el código PHP haciendo que cuente del 10 al 20.

2.1. Uso de parámetros por referencia

Hemos visto que, entre los paréntesis de las funciones, podemos incluir unos datos llamados argumentos o parámetros. Estos parámetros nos sirven para darle información a la función para poder realizar sus operaciones, como en el ejemplo anterior, donde le pasamos los dos números que tiene que sumar.

En ocasiones nos puede interesar que alguno de esos parámetros sea modificable, es decir, que la función pueda modificar directamente el valor del parámetro. En este caso, debemos pasar el parámetro por referencia, y eso en PHP se consigue anteponiéndole el símbolo & al parámetro. Por ejemplo, la siguiente función calcula la suma de los parámetros $n1 y $n2 y guarda el resultado en el parámetro $resultado, que se pasa por referencia.

```php
function sumaNumeros($n1, $n2, &$resultado)
{
    $resultado = $n1 + $n2;
}
```

A la hora de utilizar esta función, tendremos que pasarle tres parámetros, siendo el tercero de ellos una variable que poder modificar y guardar el resultado de la suma:

```php
$result = 0;
sumaNumeros(3, 30, $result);
// En este punto, la variable $result vale 33
```

> **Ejercicio 3:**
>
> Crea una página llamada **intercambia.php** en la carpeta de ejercicios. Añade dentro una función llamada `intercambia` que reciba 2 parámetros numéricos por referencia, y lo que haga sea intercambiar sus valores. Es decir, si recibe el parámetro `$a` y el `$b`, debe hacer que `$a` tome el valor de `$b`, y `$b` tome el valor de `$a`.

2.2. Uso de parámetros por defecto

También podemos incluir en la función parámetros con valores por defecto, de forma que, si no los ponemos, automáticamente toman el valor por defecto que hayamos indicado.

Estos parámetros deben ponerse todos al final, tras los parámetros "obligatorios", y debemos tener cuidado cuando no rellenamos alguno de ellos, para seguir el orden en que están. Por ejemplo, la siguiente función tiene los parámetros del nombre del alumno, su teléfono, una nota del examen y un año de nacimiento. La nota del examen tiene el valor por defecto de 0, y el año de nacimiento, por defecto es 1995.

```php
function datosAlumno($nombre, $telefono, $nota = 0, $aNacimiento = 1995)
{
    ...
}
```

Si utilizamos esta función, tenemos varias alternativas: la primera es utilizarla rellenando todos los parámetros:

```php
datosAlumno("Nacho Iborra", "611223344", 8, 1978);
```

La segunda es utilizarla dejando vacío algún parámetro que tenga valor por defecto. En este caso, si no ponemos la nota pero sí el año, entonces el año se asignaría a la nota, porque está en tercera posición en los

paréntesis:

```
datosAlumno("Nacho Iborra", "611223344", 1978);
// En este caso, la nota sería 1978, y el año de nacimiento 1995
```

En cambio, si rellenamos la nota, y dejamos vacío el año, entonces el año toma su valor por defecto.

```
datosAlumno("Nacho Iborra", "611223344", 8);
// En este caso, la nota sería 8 y el año de nacimiento 1995
```

En definitiva, si dejamos vacío algún parámetro que no esté al final, los de la derecha ocupan su lugar y se asignan a datos equivocados. En estos casos es mejor dejar vacíos todos, o rellenarlos con un valor por defecto para no descuadrar los demás.

3. Retorno de valores

Una función, además de hacer operaciones, y poder modificar parámetros, puede devolver un resultado de una operación (que puede ser matemática, una comprobación, etc.). Para hacer que una función devuelva un resultado, se utiliza la palabra `return` seguida del dato a devolver. Por ejemplo, la siguiente función devuelve la suma de los dos números que recibe como parámetros.

```
function sumaNumeros($a, $b)
{
    return ($a+$b);
}
```

Después, desde el código PHP, normalmente asignaremos lo que devuelve la función a alguna variable o expresión:

```
$resultado = sumaNumeros(3, 10);
// Aquí, la variable $resultado valdría 13
```

> **Ejercicio 4:**
>
> Crea una página llamada **descuento.php** en la carpeta de ejercicios. Crea una función llamada `calculaDescuento` que reciba un parámetro `$precio` con el precio de una compra, y un parámetro opcional llamado `$descuento` con el porcentaje de descuento a aplicar. Si no se pone este segundo parámetro, el valor por defecto será 0. La función devolverá con un *return* el precio con el

> descuento aplicado. Utiliza después la función desde el código PHP para calcular el descuento de un precio de 250 euros con un 10% de descuento, y el de un precio de 85 euros sin indicar descuento.

3.1. *Type hinting*

A pesar de que PHP es un lenguaje débilmente tipado (es decir, no tenemos que especificar de qué tipo de dato son las variables y parámetros que utilizamos), sí que permite indicar el tipo de dato en los parámetros y tipo de retorno de las funciones, si lo queremos, para "obligar" a que los datos que se pasen y recojan sean de esos tipos.

```
// Comprueba si dos números son iguales
function iguales(int $param1, int $param2): boolean
{
    return $param1 === $param2;
}
```

Puedes obtener más información al respecto aquí.

3.2. Funciones anónimas

Igual que ocurre con otros lenguajes, como JavaScript, podemos definir funciones anónimas. Es decir, funciones sin nombre, que pueden asignarse a una variable, o utilizarse en un punto determinado del programa para invocarlas directamente.

El siguiente ejemplo utiliza la función *array_filter* de PHP, que hace un filtrado de los elementos de un array. Como segundo parámetro, usamos una función anónima que especifica el criterio de filtrado para cada dato `$n` del array: nos quedamos con los números que sean múltiplos de 5:

```
$numeros = [12, 18, 5, 11, 10, 95, 3];

$multiplos5 = array_filter($numeros, function($n) {
    return $n % 5 == 0;
});

// $multiplos5 = [5, 10, 95]
```

4. Modularizando el código

La definición de funciones en PHP es una herramienta poderosa, como en cualquier otro lenguaje, porque nos permite reutilizar un mismo código en varios puntos de la aplicación, invocando varias veces la misma función. Pero... ¿qué ocurriría si queremos invocar una misma función en diferentes páginas PHP de nuestra

aplicación? Con lo que hemos visto hasta ahora, tendríamos que volver a definir la misma función en cada página. Afortunadamente, esto no tiene que ser así.

PHP ofrece un mecanismo de inclusión de contenidos en otras páginas PHP a través de cuatro directivas:

- **include fichero**: incluye el contenido del fichero indicado en el actual, en el punto donde se invoca esta directiva. Es básicamente como un *copiar-pegar* del fichero indicado en el punto de inclusión, pero sin repetir el código de nuevo.
- **include_once fichero**: similar al anterior, pero sólo lo incluye si no ha sido incluido ya anteriormente en el fichero (para evitar repetir inclusiones)
- **require fichero**: similar a include, pero emite un error fatal si no se encuentra el fichero indicado (mientras que *include* sólo emite un *warning*, pero sigue cargando la página). Todo dependerá de la importancia del fichero que estemos incluyendo.
- **require_once fichero**: similar al anterior, pero sólo carga el fichero si no ha sido incluido anteriormente.

Podemos utilizar estas directivas para varios fines:

- Definir un mismo contenido común a varias páginas (por ejemplo, mismo encabezado y/o mismo pie de página) e incluirlas en las distintas páginas que lo compartan. Aquí podemos elegir si utilizar *include* o *require*, dependiendo de lo vital que sea el contenido que estemos cargando para el correcto funcionamiento de la página.
- Definir un conjunto de funciones a compartir por varias partes de la aplicación, e incluirlos en las páginas que los necesiten (en este caso, lo normal sería utilizar *require* o *require_once*, ya que de lo contrario intentaríamos usar funciones que no existirían).

Los ficheros incluidos tienen a menudo una extensión especial, que puede ser **.inc** o bien **.inc.php**, para saber que no son fichero autónomos, sino que necesitan ser incluidos en otros.

Por ejemplo, podríamos definir un fichero llamado **funciones.inc** con algunas funciones de uso común:

```
<?php

function saluda()
{
    echo "Hola, buenas";
}

function suma($a, $b)
{
    echo a+b;
}
?>
```

Y después, utilizar este fichero en las páginas que lo necesiten, de este modo:

```
<?php
include "funciones.inc";
?>
<!DOCTYPE html>
...
<?php
suma(3, 5);
?>
...
```

> **Ejercicio 5:**
>
> Crea un archivo llamado **funciones.inc** en la carpeta de ejercicios de la sesión y añade dentro las funciones que hemos definido en los ejercicios anteriores. Incluye el fichero en cada uno de los ejercicios para que pueda hacer uso de las funciones que necesite.

5. Algunas funciones útiles predefinidas en PHP

PHP dispone de multitud de funciones para diferentes propósitos: manejo de textos, conexión con bases de datos, uso de expresiones regulares, etc. Veremos aquí algunos de estos conjuntos de funciones (salvo el de acceso a base de datos, que lo veremos con más detalle más adelante).

5.1. Funciones para manejo de textos

Veamos algunas funciones predefinidas que pueden sernos útiles para manejar datos de texto, como por ejemplo los que nos pueda enviar el usuario desde un formulario:

- La función `trim(cadena)` elimina espacios del principio y del final de la cadena, incluyendo tabulaciones o saltos de línea. Es especialmente útil para borrar espacios que se han quedado puestos accidentalmente.

```
$texto = "   Esto es un texto con espacios   ";
$texto2 = trim($texto);
// En este punto, $texto2 vale "Esto es un texto con espacios"
```

- La función `number_format(numero, decimales)` crea un texto mostrando el número indicado, con los decimales que se digan.

```
$numero = 3.14159;
$texto = number_format($numero, 2);
// En este punto, $texto vale 3.14
```

- La función `htmlspecialchars(cadena)` formatea la cadena sustituyendo algunos símbolos especiales (como por ejemplo `&`, `"`, `<` ...) por sus correspondientes códigos HTML (`&`, `&quot;`, `<` ...)

```
$texto = "if (var1 < var2 && ...";
$textoHTML = htmlspecialchars($texto);
// En este punto, $textoHTML vale "if (var1 &lt; var2 &amp;&amp; ..."
```

- Las funciones `strtoupper(cadena)` y `strtolower(cadena)` convierten toda la cadena a mayúsculas y minúsculas, respectivamente.

```
$texto = "Hola, buenas Tardes";
$textoMayus = strtoupper($texto);
// En este punto, $textoMayus vale "HOLA, BUENAS TARDES"
$textoMinus = strtolower($texto);
// En este punto, $textoMinus vale "hola, buenas tardes"
```

- La función `explode(separador, cadena)` devuelve una lista (array) con todas las partes de la cadena en que se puede dividir cortando por el separador indicado. La función `implode(separador, array)` combina todos los elementos de la lista (array), uniéndolos con el separador indicado.

```
$texto = "uno-dos-tres-cuatro";
$partes = explode('-', $texto);
// En este punto, $partes es una lista o array. En su posición 0 está
// la palabra 'uno', en la 1 está 'dos', en la 2 está '3' y en la 3
// está 'cuatro'
$texto2 = implode(';', $partes);
// Aquí $texto2 vale 'uno;dos;tres;cuatro'
```

- La función `substr(cadena, comienzo, fin)` obtiene una subcadena de la cadena, a partir de la posición de comienzo indicada hasta (opcionalmente) la posición de fin indicada (si no se pone fin, se toma hasta el final de la cadena).

```
$texto = "Hola, buenas tardes";
$texto2 = substr($texto, 6, 11);
// Aquí $texto2 vale 'buenas'
```

- La función `strcmp(cadena1, cadena2)` compara las dos cadenas viendo cuál es mayor alfabéticamente. Dará un número positivo si la *cadena1* es mayor, negativo si la *cadena2* es mayor, o cero si son iguales. La función `strcasecmp(cadena1, cadena2)` funciona como la anterior, pero sin

distinguir mayúsculas y minúsculas (las mayúsculas, si no se indica lo contrario, se consideran menores que las minúsculas).

```
$texto1 = "hola";
$texto2 = "adiós";
$resultado = strcmp($texto1, $texto2);
// Aquí $resultado es > 0, porque "hola" es mayor que "adiós".
```

- La función `strlen(cadena)` indica cuántos caracteres tiene la cadena.

```
$texto = "Hola, buenas tardes";
$caracteres = strlen($texto);
// Aquí $caracteres valdrá 19
```

- La función `strpos(cadena, parte)` indica en qué posición está la primera ocurrencia de la cadena *parte* en la cadena *cadena* (o un número negativo si no se encuentra).

```
$texto = "Hola, buenas tardes";
$posicion = strpos($texto, "buenas");
// Aquí $posicion valdrá 6
```

- La función `str_replace(viejacadena, nuevacadena , cadena)` reemplaza todas las ocurrencias de la cadena *viejacadena* por la cadena *nuevacadena* en la cadena global *cadena*.

```
$errores = "Un téxto de prueba con acento en la palabra téxto";
$corregido = str_replace("téxto", "texto", $errores);
// Aquí corregido vale "Un texto de prueba con acento en la palabra texto"
```

Ejercicio 6:

Crea una página llamada **manejo_textos.php**. Realiza en ella las siguientes operaciones:

- Crea una variable `$radio` con un radio de circunferencia (el que quieras).
- Crea una variable `$area` y calcula en ella el área del círculo, como has hecho en algún ejercicio anterior (PI * radio, definiendo la constante PI).
- Crea una variable `$textoResultado` que diga "El área calculada del círculo es" y luego ponga la variable `$area`, mostrando sólo 2 decimales (utiliza la función `number_format`). Muestra luego esta variable por pantalla con un *echo*.
- Crea una variable `$textoResultadoMayus` que convierta el texto anterior a mayúsculas, usando la función `strtoupper`. Muestra también esta variable por pantalla.

2

- Crea una variable llamada `$textoResultadoModificado` que reemplace la palabra "calculada" por la palabra "obtenida", usando la función `str_replace`, en la variable `$textoResultado`.
- Averigua la longitud del texto de la variable anterior usando la función `strlen`.
- Averigua en qué posición del texto de la variable anterior se encuentra la palabra "círculo", usando la función `strpos`.
- Crea una variable `$numeros` que tenga el valor "1,2,3,4,5", y utiliza la función `explode` para quedarte con los números por separado. Sácalos por pantalla, separados por el signo "+" ("1+2+3+4+5"), y después, intenta sumarlos entre sí y mostrar el resultado de la suma a continuación (al final, te quedará algo como "1+2+3+4+5=15").

5.2. Funciones para manejo de fechas y horas

Existen algunas funciones que pueden sernos muy útiles para manejar fechas y horas, y poderlas procesar o almacenar correctamente en bases de datos.

- `time()`: nos da el nº de segundos que han transcurrido desde el 1/1/1970. Esto nos servirá para establecer marcas temporales (*timestamps*), o para convertir después esta marca temporal en una fecha y hora concretas.
- `checkdate(mes, dia, año)` comprueba si la fecha formada por el mes, día y año que recibe como parámetros es correcta o no (da un valor de verdadero o falso)
- `date(formato, fecha)` obtiene una cadena de texto formateando la fecha con el formato indicado. Si no se indica ninguna fecha, se le aplicará el formato indicado a la fecha actual. Dentro del formato, podemos usar diferentes patrones, dependiendo del tipo de formato que queramos. Usaremos los símbolos d/D (para día numérico o con letra), m/M (para mes numérico o abreviado), y/Y (año de dos o cuatro dígitos), h/H (hora de 12 o 24 horas), i (minutos), s(segundos), y otras variantes que se pueden consultar en la documentación oficial

```
$fechaActual = time();
$textoFecha = date("d/m/Y H:i:s");
// Suponiendo que $fechaActual sea, por ejemplo, el 3 de noviembre de 2014
// a las 18:23:55, entonces $textoFecha sería '03/11/2014 18:23:55'
$esCorrecta = checkdate(15, 11, 2014);
// La variable $esCorrecta sería FALSE, porque 15 no es un mes válido
```

- `strtotime(texto)` convierte un texto que intenta representar una fecha en una fecha determinada. La fecha se representa en formato *mes/día/año* o *mes-día-año*, normalmente. Esta fecha sería incorrecta porque no existe el mes 21:

```
$fecha = strtotime("21/12/2013");
```

> **Ejercicio 7:**

> Crea una página llamada **ahora.php** en tu carpeta de ejercicios. Saca en la página la fecha y hora actuales en un formato como este: *07 Mar 2022 - 21:55:12*

5.3. Funciones para gestión de expresiones regulares

Al igual que otros lenguajes como JavaScript, desde PHP también podemos utilizar expresiones regulares para procesar ciertos textos y extraer patrones de ellos. La sintaxis de las expresiones regulares es muy similar a la de JavaScript, y se basa en un conjunto de símbolos y expresiones que podemos utilizar para representar los distintos tipos de caracteres de un texto (números, letras, espacios, etc.).

Símbolo	Significado
`^`	Se utiliza para indicar el comienzo de un texto o línea
`$`	Se utiliza para indicar el final de un texto o línea
`*`	Indica que el carácter anterior puede aparecer 0 o más veces
`+`	Indica que el carácter anterior puede aparecer 1 o más veces
`?`	Indica que el carácter anterior puede aparecer 0 o 1 vez
`.`	Representa a cualquier carácter, salvo el salto de línea
`x|y`	Indica que puede haber un elemento x o y
`{n}`	Indica que el carácter anterior debe aparecer n veces
`{m, n}`	Indica que el carácter anterior debe aparecer entre m y n veces
`[abc]`	Cualquiera de los caracteres entre corchetes. Podemos especificar rangos con un guión, como por ejemplo `[A-L]`
`[^abc]`	Cualquier carácter que no esté entre los corchetes
`\b`	Un límite de palabra (espacio, salto de línea...)
`\B`	Cualquier cosa que no sea un límite de palabra
`\d`	Un dígito (equivaldría al intervalo `[0-9]`)
`\D`	Cualquier cosa que no sea un dígito (equivaldría a `[^0-9]`)
`\n`	Salto de línea
`\s`	Cualquier carácter separador (espacio, tabulación, salto de página o de línea)
`\S`	Cualquier carácter que no sea separador
`\t`	Tabulación
`\w`	Cualquier alfanumérico incluyendo subrayado (igual a `[A-Za-z0-9_]`)
`\W`	Cualquier no alfanumérico ni subrayado (`[^A-Za-z0-9_]`)

Algunos símbolos especiales (como el punto, o el +), se representan literalmente anteponiéndoles la barra invertida. Por ejemplo, si queremos indicar el carácter del punto, se pondría `\.` Veamos algunos ejemplos:

Ejemplo	Significado
`^\d{9,11}$`	Entre 9 y 11 dígitos
`^\d{1,2}\/\d{1,2}\/\d{2,4}$`	Fecha (d/m/a)

A partir de esa sintaxis, disponemos de las siguientes funciones para procesar expresiones regulares:

- `preg_match(patron, texto)` devuelve 1 si el texto encaja en el patrón

```
$dni = "11223344A";
if (preg_match("/^\d{9}[A-Z]$/", $dni) == 1)
{
    echo "DNI válido";
}
```

- `preg_replace(patron, reemplazo, cadena)` examina la cadena buscando coincidencias del patrón, y reemplaza el texto encontrado por el texto de reemplazo.
- `preg_split(patron, cadena, limite)` devuelve un array o lista de cadenas, resultado de todas las ocurrencias del patrón en la cadena. Podemos especificar opcionalmente un límite de cadenas devueltas.

Además, podemos trabajar con expresiones regulares formando **grupos**. Esto se refiere a que, en un patrón de expresión regular podemos definir partes de esa expresión entre paréntesis, formando lo que se llaman grupos, cuando queremos que, a la vez que se encuentran coincidencias del patrón, se extraigan partes del mismo con información.

Por ejemplo, la siguiente variable tiene una fecha. Utilizamos una expresión regular para extraer el día, mes y año por separado, creando grupos para cada cosa. Como resultado, se obtiene una lista o array donde, en la primera posición, se tiene la fecha completa detectada, y en las siguientes 3 posiciones se tienen los tres grupos identificados (día, mes y año, respectivamente).

```
$fecha = "03-11-2014";
if (preg_match("/^([0-9]{2})-([0-9]{2})-([0-9]{4})$/", $fecha, $partes) == 1)
{
    echo "La fecha completa es " . $partes[0];
    echo "El día es " . $partes[1];
    echo "El mes es " . $partes[2];
    echo "El año es " . $partes[3];
} else {
    echo "Formato de fecha no válido";
}
```

> **Ejercicio 8:**
>
> Crea una página llamada **comprueba_hora.php** en tu carpeta de ejercicios. Crea una variable de texto con una hora en ella (por ejemplo, "21:30:12"), y luego procésala para extraer por separado la hora, el minuto y el segundo, y comprobar si es una hora válida. Por ejemplo, la hora anterior sí debería ser válida, pero si ponemos "12:63:11" no debería serlo, porque 63 no es un minuto válido.

5.4. Funciones para manejo de ficheros

A veces nos puede resultar útil leer un fichero de texto o escribir información en él. Existen multitud de funciones en PHP para abrir un fichero, leerlo línea a línea, o leer o guardar un conjunto de bytes... Vamos a ver aquí sólo algunas de las funciones más útiles para manejo de ficheros.

- `readfile(fichero)` lee un fichero entero y lo vuelca al buffer de salida (es decir, a la página que se está generando, si estamos en una página PHP).
- `file(fichero)` lee un fichero entero y devuelve un array o lista, donde en cada posición hay una línea del fichero
- `file_get_contents(fichero)` lee un fichero entero y lo devuelve en una cadena (no en un array, como la anterior)
- `file_put_contents(fichero, texto)` escribe la cadena de texto en el fichero indicado, sobreescribiendo su anterior contenido si lo tenía. En el caso de que no queramos sobreescribir, sino añadir, pondremos un tercer parámetro con la constante FILE_APPEND.
- `file_exists(fichero)` devuelve TRUE o FALSE dependiendo de si el fichero indicado existe o no
- `filesize(fichero)` devuelve el tamaño en bytes del fichero, o FALSE si no existe

El siguiente ejemplo lee un archivo llamado *libro.txt* (en la misma carpeta que el archivo actual) añade la palabra "Fin" al final y vuelve a guardar su contenido.

```
$fichero = "libro.txt";
if (file_exists($fichero))
{
    $contenido = file_get_contents($fichero);
    $contenido .= "Fin";
    file_put_contents($fichero, $contenido);
}
```

Este mismo ejemplo lo podríamos haber hecho así también, añadiendo la nueva palabra Fin al final de lo existente:

```php
$fichero = "libro.txt";
if (file_exists($fichero))
{
      file_put_contents($fichero, "Fin", FILE_APPEND);
}
```

Ejercicio 9:

Crea una página llamada **copia_seguridad.php**. En la misma carpeta, crea un archivo llamado *datos.txt* con tus datos personales en varias líneas (Nombre, Dirección, Teléfono y E-mail, tuyos o inventados). Haz que la página php lea el archivo y lo guarde en otro llamado *copia_datos.txt*, en la misma carpeta.