Uso básico de PHP

Implantación de Aplicaciones Web

Nacho Iborra

IES San Vicente

Introducción a PHP

PHP es un lenguaje de *scripting* diseñado para desarrollar aplicaciones web en el lado del servidor. Fue concebido en el año 1994 para uso más o menos personal, pero su gran aceptación y las mejoras que se le fueron añadiendo con posterioridad han hecho que ya en 2005 se encontrara instalado en más de 23 millones de dominios.

Podemos encontrar gran variedad de sitios web realizados con PHP, normalmente sobre un servidor Apache: tiendas virtuales en las que poder crearnos un carro de la compra y pagar sobre una plataforma de pago de una entidad bancaria, cursos virtuales en plataformas Moodle (CMS hecho en PHP), o páginas web más o menos complejas realizadas directamente en PHP, o con algún gestor de contenidos en PHP como Wordpress o Joomla.

Características principales

Entre sus principales características, aparte de su popularidad, podemos citar las siguientes:

- Es ***open source***, por lo que podemos acceder a su código y usarlo, modificarlo o distribuirlo.
- Es **fácil de aprender**, si se compara con otros lenguajes de programación como C o Perl.
- Es **multiplataforma**, pues se puede ejecutar en servidores web (como Apache o IIS) en diversos sistemas operativos. Esta es una diferencia importante con otras tecnologías como .NET (exclusiva de Windows), aunque hay otros lenguajes de servidor que también son multiplataforma, como Java/JSP.
- El código que escribamos está **embebido en páginas HTML**. Veremos más adelante cómo las instrucciones de PHP se colocan entre las etiquetas propias de HTML.
- Admite **programación orientada a objetos** desde su versión 5, con lo que podemos trabajar con objetos y clases.
- Permite **gestionar errores mediante excepciones**.
- Permite **procesar archivos XML**
- Permite acceder a diversos **sistemas de bases de datos** (Informix, MySQL, Oracle, PostgreSQL...)
- Permite conectar con **otros servicios** (correo, FTP, etc.)
- Admite diversas librerías y entornos para implementar sistemas de **comercio electrónico.**

Uso básico de PHP

La sintaxis básica para escribir código PHP consiste en un bloque como el siguiente:

```
<?php ... código PHP ... ?>
```

Dentro de los símbolos *<?php* (de apertura) y *?>* (de cierre) pondremos las instrucciones que consideremos. Todo esto puede ocupar una sola línea (para instrucciones simples y cortas) o dividir el código en varias líneas, una por instrucción.

Debemos tener en cuenta que:

- Cada instrucción en PHP termina siempre en un punto y coma (;) que la separa de la siguiente, aunque estén en líneas separadas.
- Los **comentarios** en PHP se pueden poner de diversas formas.

  ```
  /* Comentario que puede ocupar varias líneas */
  // Comentario de una línea
  # Otro ejemplo de comentario de una línea
  ```

  Estos comentarios ayudan a comprender mejor el código que escribamos, o a veces a delimitar fragmentos de código y separarlos del resto de la página.

Veamos a continuación un ejemplo sencillo de código PHP, embebido dentro de una página HTML. En este caso, creamos unas variables en PHP que almacenan un nombre y un año, y luego mostramos esas variables entre el contenido HTML propiamente dicho:

```
<html>
<head>
      ...
</head>
<body>
      <h1>Página de prueba en PHP</h1>
      <?php
            // Variables para almacenar el nombre y el año actual
            $nombre = "Nacho Iborra";
            $anyo = 2014;
      ?>
      <p>El autor de esta página es <?php echo $nombre; ?> y está realizada en
      el año <?php echo $anyo; ?>.</p>
...
```

Observa cómo podemos incluir código PHP en cualquier zona de la página. En la primera, hemos definido dos variables, *nombre* y *anyo*, y después, con instrucciones cortas, las hemos mostrado en otras zonas de la página (instrucción *echo*).

Lo que hace el servidor cuando se le solicita esta página es procesarla, detectar el código PHP, ejecutarlo y sustituirlo en la página por el código HTML que éste genera (en el ejemplo anterior, mostrar el nombre y el año de nacimiento de las variables, en los lugares correspondientes).

Opcionalmente, en el caso de que sólo tengamos instrucciones *echo*, podemos sustituir la estructura *<?php ... ?>* por la estructura *<?= ... ?>* y ahorrarnos la instrucción *echo*. Así, el párrafo que se muestra en el ejemplo anterior lo podríamos poner así:

```
<p>El autor de esta página es <?= $nombre; ?> y está realizada en
el año <?= $anyo; ?>.</p>
```

También podemos utilizar indistintamente la instrucción **print** en lugar de *echo* para mostrar información por pantalla.

```
<p>El autor de esta página es <?php print $nombre; ?> y está realizada en el año <?php print $anyo; ?>.</p>
```

Prueba de páginas PHP

Para que el código PHP sea reconocido, debemos guardar las páginas web con extensión **.php** en lugar de *.html*. Después, debemos colocarlas en alguna carpeta dentro de la carpeta de aplicaciones web de nuestro servidor web (en nuestro caso, Apache). Es decir, en la carpeta *htdocs* o alguna subcarpeta (en Windows) o */var/www* (en Linux), si no hemos cambiado la configuración por defecto. Finalmente, deberemos poner en marcha el servidor Apache si no lo está, y acceder a la página correspondiente escribiendo su URL. Por ejemplo, si guardamos el ejemplo anterior en el archivo *ejemplo.php* dentro de la carpeta *htdocs/ejemplos*, entonces podríamos probar el ejemplo (suponiendo que Apache está escuchando peticiones en el puerto 80) con:

```
http://localhost/ejemplos/ejemplo.php
```

En el caso de que Apache esté configurado en otro puerto que no sea el puerto por defecto (por ejemplo, el 8080), tendríamos que modificar la URL, poniendo el número de puerto detrás de *localhost*, separados por dos puntos:

```
http://localhost:8080/ejemplos/ejemplo.php
```

Podemos observar al cargar la página cómo desde el navegador (cliente) **no se ve nada del código PHP**, sino el resultado que dicho código genera (en este caso, el código HTML que muestra el nombre y el año).

Ejercicio 1

Crea una carpeta llamada **ejercicios** en tu carpeta de documentos de Apache. En esta carpeta guardarás este ejercicio y los siguientes, ya que serán muchos y así evitamos llenar la carpeta de documentos de demasiadas subcarpetas con ejercicios cortos.

Para este ejercicio, crea un documento en esta carpeta llamado **info_basica.php**, similar al del ejemplo anterior, pero mostrando tu nombre y tu año de nacimiento. Es decir, crearás dos variables para almacenar estos dos datos, y los mostrarás en una frase que diga "*Me llamo XXXX y nací en el año YYYY*". Prueba la página en un navegador y echa un vistazo al código fuente, intentando detectar qué contenidos HTML se han generado desde PHP.

Variables y tipos de datos

Variables

Como en todo lenguaje de programación, las variables en PHP nos van a servir para almacenar información, de manera que, además de tenerla disponible, podemos modificarla o cambiarla por otra durante el tiempo de ejecución de la aplicación web.

Como hemos visto en el ejemplo anterior, las variables en PHP se definen mediante el símbolo del dólar ($) seguido de una serie de letras, números o caracteres de subrayado, aunque no pueden empezar por número. Ejemplos de nombres de variables válidos son: *$nombre, $primer_apellido, $apellido2*... En las variables, se distinguen mayúsculas de minúsculas, y no hace falta declararlas (es decir, no se indica de qué tipo son, como en lenguajes como C o Java, ni se les reserva memoria de antemano).

Veamos algunos ejemplos de uso de variables:

```
<?php
      $edad = 36;
      $nombre = "Nacho";
      ...
      echo $nombre;
      echo $edad;
?>
```

Comprobar el estado de las variables

Es posible que, en algún momento de la ejecución del programa, una variable no tenga un valor definido, o queramos eliminar el valor que tiene. Para ello tenemos algunas funciones útiles:

- **unset($variable)** permite borrar la variable (como si no la hubiéramos creado)
- **isset($variable)** permite comprobar si una variable existe
- **empty($variable)** permite comprobar si una variable está vacía, es decir, no tiene un valor concreto asignado.

```
<?php
      $dato="Hola";
      unset($dato);           // La variable dato deja de existir
      $dato = "";
      echo empty($dato);      // Diría que es cierto, porque $dato está vacía
?>
```

Tipos de datos

Las variables almacenan datos, y esos datos son de un tipo concreto. Por ejemplo, pueden ser números, o textos. En concreto, PHP soporta los siguientes tipos de datos básicos:

- **Booleanos**: datos que sólo pueden valer *verdadero* o *falso* (en PHP se representan con las palabras *TRUE* y *FALSE*, respectivamente, en mayúsculas).

- **Enteros**: números sin decimales. Los podemos representar en formato decimal (el normal, por ejemplo 79), formato octal (poniendo un 0 delante, por ejemplo 0233) o hexadecimal (poniendo 0x delante, por ejemplo 0x1A3).
- **Reales**: números con decimales. La parte decimal queda separada de la entera con un punto. Por ejemplo, 3.14. También podemos usar notación científica, como por ejemplo 1.3e3, que equivale a $1.3 \cdot 10^3$.
- **Cadenas de texto**: se pueden representar entre comillas simples o dobles. Si los representamos con comillas dobles, podemos intercalar variables dentro.

Veamos algunos ejemplos de cada tipo:

```
<?php
      $edad = 20;
      $esMayorEdad = TRUE;
      $nota = 9.5;
      $nombre = "Juan Pérez";

      echo "El alumno $nombre tiene una nota de $nota";
?>
```

Además, en PHP podemos manejar otros tipos de datos algo más complejos, que son:

- **Arrays**: conjuntos de datos
- **Object**: para programación orientada a objetos, almacena instancias de clases
- **Resource**: para recursos externos, como una conexión a una base de datos
- **null**: es un valor especial para darle a variables que, en un momento dado, no tengan un valor concreto. Así, quedan vacías, sin valor.

Veremos algunos de estos tipos complejos más adelante.

Conversiones entre tipos de datos

Si tenemos un dato de un tipo y lo queremos convertir a otro, se tienen una serie de funciones que nos permiten hacer estas conversiones.

- **intval** sirve para convertir un valor (en formato cadena, o real) a entero.
- **doubleval** sirve para convertir un valor a número real.
- **strval** sirve para convertir un valor a una cadena de texto.

Es habitual usarlas cuando le pedimos datos al usuario en un formulario. Si por ejemplo le pedimos que escriba su edad en un cuadro de texto, este valor se envía como tipo cadena de texto, no como un número, y luego tendríamos que convertirlo a número entero. Realmente, estas conversiones son automáticas con las últimas versiones de PHP, pero conviene saber que estas funciones existen para poderlas utilizar si es el caso. Veamos otro ejemplo más sencillo, con una variable cadena de texto que convertimos al entero correspondiente:

```
<?php
      $textoEdad='21';
      $edad = intval($textoEdad);
?>
```

Constantes

Hemos visto que las variables son datos cuyo valor puede cambiar a lo largo de la ejecución de la aplicación. A veces nos puede interesar almacenar otros datos que no queramos que cambien a lo largo del programa. Por ejemplo, si almacenamos el número *pi*, ese valor siempre va a ser el mismo, no lo vamos a cambiar.

Para definir constantes en PHP se utiliza la función **define**, y entre paréntesis pondremos el nombre que le damos a la constante (entre comillas), y el valor que va a tener, separados ambos datos por coma. Después, para utilizar la constante más adelante, usamos el nombre que le hemos dado, pero sin las comillas. Veamos este ejemplo que calcula la longitud de una circunferencia:

```
<?php
      define('PI', 3.1416);
      $radio = 5;
      $longitud = 2 * PI * radio;

      echo "La longitud de la circunferencia es $longitud";
?>
```

Aunque no es obligatorio, sí es bastante convencional que las constantes tengan todo su nombre en mayúsculas, para distinguirlas a simple vista de las variables (aunque, además, las variables en PHP empiezan por un dólar, y las constantes no).

Ejercicio 2

Crea una página en la carpeta de *ejercicios* llamada **area_circulo.php**. En ella, crea una variable *$radio* y ponle el valor 3.5. Según esa variable, calcula en otra variable el área del círculo (PI * radio^2, deberás definir la constante PI), y muestra por pantalla el texto "El área del círculo es XX.XX", donde XX.XX será el resultado de calcular el área.

Operaciones

Podemos realizar distintos tipos de operaciones con los datos que manejamos en PHP: aritméticas, comparaciones, asignaciones, etc. Veremos los operadores que podemos utilizar en cada caso.

Operaciones aritméticas

Son las operaciones matemáticas básicas (sumar, restar, multiplicar...). Los operadores para llevarlas a cabo son:

+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto de división
.	Concatenación (para textos)
++	Autoincremento
--	Autodecremento

El operador . sirve para concatenar o enlazar textos, de forma que podemos unir varios en una variable o al sacarlos por pantalla:

```
$edad = 36;
$nombre = "Nacho";
$texto = "Hola, me llamo " . $nombre . " y tengo " . $edad . " años.";
// En este punto, $texto vale "Hola, me llamo Nacho y tengo 36 años."
echo "El usuario se llama " . $nombre;
```

Por su parte, los operadores de autoincremento y autodecremento, aumentan o disminuyen en 1 el valor de la variable a la que acompañan:

```
$numero = 3;
$numero++;
// En este punto, $numero vale 4
// Es lo mismo que haber hecho $numero = $numero + 1
```

Operaciones de asignación

Permiten asignar a una variable un cierto valor. Se tienen los siguientes operadores:

=	Asignación simple
+=	Autosuma
-=	Autoresta
*=	Automultiplicación
/=	Autodivisión

%=	Autoresto
.=	Autoconcatenación

La asignación simple ya la hemos visto en ejemplos previos, y sirve simplemente para darle un valor a una variable.

```
$dato = 3;
```

Los operadores de *Auto...* afectan a la propia variable, sumándole/restándole/... etc. un valor. Por ejemplo, si hacemos algo como:

```
$dato *= 5;
```

estamos multiplicando la propia variable *$dato* por 5, y guardando el resultado en la misma variable *$dato*. Así, si antes valía 3, ahora pasará a valer 15.

Operaciones de comparación

Estas operaciones permiten comparar valores entre sí, para ver cuál es mayor, o si son iguales, entre otras cosas. En concreto, los operadores disponibles son:

==	Igual que
===	Idéntico a (igual y del mismo tipo)
!= , <>	Distinto de
!==	No idéntico
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

Lo que se obtiene con estas comparaciones es un valor booleano (es decir, verdadero o falso). Por ejemplo, 5 != 3 sería verdadero, y 4 <= 1 sería falso.

Operaciones lógicas

Estas operaciones permiten enlazar varias comprobaciones simples, o cambiarles el sentido, según el operador. En concreto tenemos estos operadores lógicos en PHP:

and , &&	Operador AND (Y)
or , ||	Operador OR (O)
xor	Operador XOR
!	Operador de negación

- El operador AND (bien con *and*, bien con &&) permite enlazar dos comprobaciones, formando una sola que será cierta si las dos comprobaciones lo son. Por ejemplo, la siguiente comprobación es cierta si las variables *n1* y *n2* son positivas las dos.

  ```
  echo $n1 >= 0 && $n2 >= 0;
  ```

- El operador OR (bien con *or*, bien con ||) permite enlazar dos comprobaciones, formando una sola que será cierta si alguna de las dos comprobaciones (o las dos) lo son. Por ejemplo, la siguiente comprobación es cierta si alguna de las variables *n1* o *n2* es positiva (o las dos lo son).

```
echo $n1 >= 0 || $n2 >= 0;
```

- El operador XOR es similar al anterior, pero la comprobación conjunta es cierta sólo si una de las dos comprobaciones lo es, pero no las dos. Este ejemplo será cierto si *n1* o *n2* son positivos (pero no los dos)

```
echo $n1 >= 0 xor $n2 >= 0;
```

- El operador de negación invierte el sentido de una comprobación (si era verdadera, la vuelve falsa, y viceversa). Por ejemplo, si queremos ver si una edad no es mayor de edad, podríamos ponerlo así:

```
echo !($edad >= 18);
```

  Aunque también podríamos ponerlo así (las negaciones normalmente siempre tienen una vía alternativa para no usarse):

```
echo $edad < 18;
```

Precedencia de operadores

¿Qué ocurre si tenemos varios operadores de distintos tipos en una misma expresión? PHP sigue un orden a la hora de evaluarlos:

1. Toda expresión entre **paréntesis** (por lo que si queremos que algo se haga antes que lo demás, lo pondremos entre paréntesis)
2. Autoincrementos y autodecrementos
3. Negaciones y cambios de signo
4. Multiplicaciones, divisiones y restos
5. Sumas, restas y concatenaciones
6. Comparaciones
7. &&
8. ||
9. Asignaciones
10. and
11. xor
12. or

Existen, además, otros operadores que no hemos visto aquí, como operadores de bits, conversiones tipo *cast*, operador ternario... pero no son tan habituales ni importantes.

Ejercicio 3

Intenta predecir qué resultado va a sacar por pantalla cada instrucción *echo* de este código PHP. Luego podrás comprobar si estabas en lo cierto poniendo el código en una página y probándolo en un navegador.

```
<?php
	$num1 = 3;
	$num2 = 5;
	$num3 = 8;
	$num1 *= 4;

	echo $num1;
	echo $num1 <= $num2;
	echo $num3 > $num1 and $num3 > $num2;
	echo $num3 > $num1 or $num3 > $num2;
	echo $num1 > $num2 xor $num1 > $num3;
	$num3--;
	echo $num3;
	$num3 += $num1;
	echo $num3;
?>
```

Control de flujo

Estructuras selectivas. La instrucción if

A veces nos interesa realizar una operación si se cumple una determinada condición y no hacerla (o hacer otra distinta) si no se cumple esa condición. Por ejemplo, si está vacía una variable querremos hacer una cosa, y si no lo está, hacer otra. Para decidir entre varios caminos a seguir en función de una determinada condición, al igual que en otros lenguajes como Javascript, Java o C, se utiliza la estructura **if**:

```
if (condicion)
{
      instruccion1;
      instruccion2;
      ...
}
```

Esta estructura lleva entre paréntesis una condición (o una combinación de ellas, utilizando los operadores lógicos &&,||, and, xor, or...). Si esa condición se cumple, se ejecutarán las instrucciones que tenga entre llaves, y si no se cumple, no se ejecutarán. Por ejemplo, si nos guardáramos el e-mail que ha introducido el usuario en una variable *$email*, podríamos comprobar si está vacío, y si es así, mostrar un mensaje de error:

```
if (empty($email))
{
      echo "Debes rellenar el e-mail antes de continuar";
}
```

En el caso de que queramos hacer otra cosa cuando no se cumple la condición, tenemos que utilizar la estructura **if..else**:

```
if (condicion)
{
      instruccion1a;
      instruccion2a;
      ...
}
else
{
      instruccion1b;
      instruccion2b;
      ...
}
```

En este caso, si la condición se cumple, como antes, ejecutaremos las instrucciones que hay entre las llaves del *if*, y si no se cumple, ejecutaremos las instrucciones que hay entre las llaves del *else*. En el ejemplo anterior, podríamos mostrar un mensaje de error o uno de OK, dependiendo de si el campo del e-mail está relleno o no:

```
if (empty($email))
{
      echo "Debes rellenar el e-mail antes de continuar";
}
else
{
```

```
      echo "Todo correcto. Podemos continuar";
}
```

Si queremos elegir entre más de dos caminos, podemos utilizar la estructura **if..elseif.. elseif..** y poner una condición en cada *if* para cada camino. Por ejemplo, imaginemos que tenemos una variable *edad* donde almacenaremos la edad del usuario. Si el usuario no llega a 10 años le diremos que no tiene edad para ver la web. Si no llega a 18 años, le diremos que aún es menor de edad, pero puede ver la web, y si tiene más de 18 años le diremos que está todo correcto:

```
$edad = ...
if ($edad < 10)
{
      echo "No tienes edad para ver esta web";
}
elseif ($edad < 18)
{
      echo "Aún eres menor de edad, pero puedes acceder a esta web";
}
else
{
      echo "Todo correcto";
}
```

El último *else* podría ser *elseif ($edad >= 18)*, el efecto sería el mismo en este caso.

Ejercicio 4

Crea una página llamada **prueba_if.php** en la carpeta de ejercicios del tema. Crea en ella dos variables llamadas *$nota1* y *$nota2*, y dales el valor de dos notas de examen cualesquiera (con decimales si quieres). Después, utiliza expresiones *if..else* para determinar qué nota es la mayor de las dos.

Ejercicio 5

Modifica el ejercicio anterior añadiendo una tercera nota *$nota3*, y determinando cuál de las 3 notas es ahora la mayor. Para ello, deberás ayudarte esta vez de la estructura *if..elseif..else*.

Repeticiones

Para poder repetir código o recorrer conjuntos de datos, al igual que en otros lenguajes de programación, disponemos de la estructura **for**. Supongamos que tenemos una variable *$lista* con una lista de elementos. Si queremos recorrerla, tenemos dos opciones: la primera es utilizar una variable que vaya desde el principio de la lista (posición 0) hasta el final (indicado por la función *count*):

```
for ($i = 0; $i < count($lista); $i++)
{
      echo $lista[i];               // Muestra el valor de cada elemento
}
```

La segunda alternativa es utilizar una variable que sirva para almacenar cada elemento de la lista. Para esta opción usamos la estructura **foreach**

```
foreach ($lista as $elemento)
{
```

```
      echo $elemento;                // Muestra el valor de cada elemento
}
```

Además, existe una tercera forma de repetir un conjunto de instrucciones: la estructura **while**. Pondremos entre paréntesis una condición que debe cumplirse para que las instrucciones entre las llaves se repitan. Este código cuenta del 1 al 5:

```
$numero = 1;
while ($numero <= 5)
{
      echo $numero;
      $numero++;
}
```

Una variante de la estructura *while* es la estructura **do..while**, similar a la anterior, pero donde la condición para terminar el bucle se comprueba al final:

```
$numero = 1;
do
{
      echo $numero;
      $numero++;
} while ($numero <= 5);
```

Ejercicio 6

Crea una página llamada **contador.php** en la carpeta de ejercicios del tema. Utiliza una estructura *for* para contar los números del 1 al 100 (separados por comas), y luego una estructura *while* para contar los números del 10 al 0 (una cuenta atrás, separada por guiones). Al final debe quedarte algo como esto:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15...

10-9-8-7-6-5-4-3-2-1-0

Intercalar control de flujo y HTML

Podemos intercalar bloques de código PHP y bloques HTML, incluso cortando bloques de control de flujo para introducir el código HTML. Por ejemplo:

```
<?php
if ($edad < 10=)
{
?>
   <div class="menor">Eres demasiado pequeño para entrar a esta web</div>
<?php }  elseif ($edad < 18) {  ?>
   <div class="mediano">No eres mayor de edad, pero puedes entrar</div>
<?php } else {  ?>
   <div class="mayor">Todo correcto. Bienvenido</div>
<?php } ?>
```

Sería equivalente a haber hecho un solo bloque PHP que, con instrucciones *echo*, sacara el contenido HTML, pero a veces es más cómodo poner el HTML directamente.

Ejercicio 7

Modifica el ejercicio anterior y añádele algún *h1* y párrafos explicativos a la página, fuera del código PHP, explicando lo que se va a hacer. Por ejemplo, que te quede algo así:

Contadores

Este contador va del 1 al 100:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15...

Este otro va del 10 al 0

10-9-8-7-6-5-4-3-2-1-0

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.

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. 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.

Uso de parámetros

Igual que ocurre en otros lenguajes, como Javascript, 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 (numérico) 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 8

Crea una página llamada **saludo_funciones.php**. Crea dentro 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 9

Crea una página llamada **contador_funciones.php** en la carpeta de ejercicios del tema. 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.

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.

```
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:

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

Ejercicio 10

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.

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.

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

Si utilizamos esta función, tenemos varias alternativas:

- Utilizarla rellenando todos los parámetros:

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

- 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.

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 11

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.

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).

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 (&, ", <...)

```
$texto = "if (var1 < var2 && ...";
$textoHTML = htmlspecialchars($texto);
// En este punto, $textoHTML vale "if (var1 &lt; var2 &amp;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 mayores 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 12

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^2, 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 *$textoResultadoMayus* que convierta el texto anterior a mayúsculas, usando la función *strtoupper.* Muestra también esta variable por pantalla.
- Crea una variable llamada *$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").

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 (por ejemplo, F para nombre del mes completo, n para indicar el mes numéricamente sin ceros de relleno, etc.)

```
$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. Tiene el inconveniente de que el formato de la fecha depende del sistema e idioma del software que se esté utilizando. En algunos lugares ponen primero el mes y luego el día, en otros es al revés...

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

Ejercicio 13

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 2014 - 21:55:12*

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.).

^	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:

^\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:

- **ereg(patron, cadena, partes)** busca en la cadena la coincidencia del patrón. Si se han hecho grupos, cada uno de ellos se guarda en el array de partes
- **ereg_replace(patron, reemplazo, cadena)** examina la cadena buscando coincidencias del patrón, y reemplaza el texto encontrado por el texto de reemplazo.

- **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.

Respecto a lo que se ha comentado de los **grupos**, 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 (ereg("([0-9]{2})-([0-9]{2})-([0-9]{4})", $fecha, $partes))
{
      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 14

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.

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:

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

Ejercicio 15

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.

Tablas o arrays

Las tablas o arrays nos permiten almacenar varios datos en una sola variable, de forma que podemos acceder a esos datos utilizando distintos tipos de índices. En PHP existen dos tipos de arrays:

- **Numéricos**: la posición que ocupa cada elemento del array en la lista la indica un número, y podemos acceder a esa posición indicando ese número entre corchetes. Las posiciones empiezan a numerarse en la 0.
- **Asociativos**: la posición que ocupa cada elemento en la lista viene dada por un nombre, y podemos localizar cada elemento a través del nombre que le hemos dado.

Arrays numéricos

Estos arrays podemos crearlos de tres formas posibles:

- Indicando entre paréntesis sus elementos, y anteponiendo la palabra **array,**

```
$tabla = array('Uno', 'Dos', 'Tres');
```

- Indicando a mano el índice que queremos rellenar, y el valor que va a tener (los índices intermedios que queden sin valor se quedarán como huecos vacíos)

```
$tabla[0] = 'Uno';
$tabla[1] = 'Dos';
$tabla[2] = 'Tres';
```

- Indicando el nombre del array con corchetes vacíos cada vez que queramos añadir un elemento. Así, se añade al final de los que ya existen:

```
$tabla[] = 'Uno';
$tabla[] = 'Dos';
$tabla[] = 'Tres';
```

Después, para sacar por pantalla algún valor, o usarlo en alguna expresión, pondremos el nombre del array y, entre corchetes, la posición que queremos:

```
echo $tabla[1];          // Sacaría 'Dos' en los casos anteriores
```

Arrays asociativos

En este caso, al crear el array debemos indicar, además de cada valor, la clave que le vamos a asociar, y por la que lo podemos encontrar, separados por el símbolo "=>". Por ejemplo, este array guarda para cada nombre de alumno su nota:

```
$notas = array('Manuel García'=>8.5, 'Ana López'=>7, 'Juan Solís'=>9);
```

También podemos rellenarlo, como en el caso anterior, indicando entre corchetes cada clave, y luego su valor:

```
$notas['Manuel García'] = 8.5;
$notas['Ana López'] = 7;
...
```

Después, si queremos sacar la nota del alumno *Manuel García*, por ejemplo, pondremos algo como:

```
echo $notas['Manuel García'];
```

En este tipo de arrays no podremos usar índices numéricos, porque no hay posiciones numéricas.

Arrays multidimensionales

Los arrays creados anteriormente son unidimensionales (sólo hay una lista o fila de elementos). Pero podemos tener tantas dimensiones como queramos. Es habitual encontrarnos con arrays bidimensionales (tablas), para almacenar información. En este caso, tendremos un corchete para cada dimensión. Por ejemplo, para crear una tabla como la que tenemos a la derecha, necesitaremos un código como el siguiente:

```
$tabla[0][0] = 34.1;
$tabla[0][1] = 141;
$tabla[1][0] = 36.4;
$tabla[1][1] = 150;
$tabla[2][0] = 33.5;
$tabla[2][1] = 155;
```

34,1	141
36,4	150
33,5	155

También podemos tener arrays multidimensionales de tipo asociativo, o array mixtos (donde algunas dimensiones son numéricas y otras asociativas). Por ejemplo:

```
$tabla2 =
array(
      array('nombre' => 'Juan García',
            'dni'=> '11111111A',
            'idiomas' => array('inglés', 'valenciano', 'español')
      ),
      array('nombre' => 'Elisa Rodríguez',
            'dni' => '22222222B',
            'idiomas' => array('francés', 'español')
      )
);
```

Podríamos acceder al segundo idioma hablado por la segunda persona con:

```
echo $tabla2[1]['idiomas'][1];
```

Podemos crear igualmente estos arrays usando corchetes, definiendo lo que queremos en cada dimensión:

```
$tabla2[0]['nombre'] = 'Juan García';
$tabla2[0]['dni'] = '11111111A';
$tabla2[0]['idiomas'][0] = 'inglés';
$tabla2[0]['idiomas'][1] = 'valenciano';
...
```

Recorrido de arrays

Para recorrer arrays unidimensionales podemos utilizar la expresión *foreach*, que nos devuelve el valor de cada posición, independiente de si es un array numérico o asociativo:

```
foreach ($notas as $nota)
{
      echo $nota;
```

```
}
```

Para arrays asociativos, también podemos usar esta instrucción *foreach*, indicando en la parte derecha del *as* dos variables (una para la clave y otra para el valor):

```
foreach ($notas as $alumno=>$nota)
{
      echo "El alumno $alumno tiene un $nota";
}
```

Las funciones **list($var1, $var2...)** y **each($array)** también nos pueden ser útiles para recorrer arrays asociativos, de forma que la segunda nos devuelve la clave y el valor de cada posición del array, y la primera almacena cada cosa en una variable separada:

```
while(list($alumno, $nota) = each($notas))
{
      echo "El alumno $alumno tiene un $nota";
}
```

En el caso de arrays bidimensionales numéricos, para recorrer todos los elementos de un array como el de la tabla numérica anterior, necesitaremos un doble bucle (también llamado bucle anidado): uno que recorra las filas, y otro las columnas:

```
for ($i = 0; $i < 3; $i++)
{
      for ($j = 0; $j < 2; $j++)
      {
            echo $tabla[$i][$j];
      }
}
```

Funciones para arrays

PHP dispone de varias funciones útiles a la hora de manipular arrays. Algunas de las más habituales son:

- **count($array)** nos indica cuántos elementos tiene el array. Es útil para utilizarlo en bucles y saber cuántas repeticiones podemos hacer sobre el array.
- **sort($array)** y **rsort($array)** ordenan y reindexan un array numérico (la segunda en orden decreciente)
- **asort($array)** y **arsort($array)** ordenan y reindexan un array asociativo (la segunda en orden decreciente), por sus valores.
- **ksort($array)** y **krsort($array)** ordenan un array asociativo por sus claves (la segunda en orden decreciente).

Ejercicio 16

Crea una página llamada **tabla_multiplicar.php** en tu carpeta de ejercicios. Crea en ella un array numérico bidimensional donde en cada fila almacenes la tabla de multiplicar de un número del 0 al 9; debería quedarte un array como éste (sin sacar nada por pantalla):

0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10
0	2	4	6	8	10	12	14	16	18	20
…	…	…	…	…	…	…	…	…	…	…

Después, recorre la tabla mostrando en la página la tabla de multiplicar de cada número, de forma que te quede algo así:

Tabla del 0:

0 x 1 = 0

0 x 2 = 0

...

Tabla del 1:

1 x 0 = 0

...

Control de errores

En ocasiones podemos hacer operaciones, o utilizar funciones, que pueden provocar un error grave en la aplicación. Por ejemplo, una división por cero, o una lectura de un fichero que no existe. Si no controlamos esos errores, se puede "disparar" un mensaje de error en el programa que muestre su mal funcionamiento, o lo que es peor, que revele algún dato privado, como la ubicación de un servidor, la ruta hacia un archivo, o algún nombre de usuario o contraseña. Para evitar que ciertas operaciones que puedan causar errores alteren de esa forma el funcionamiento de la aplicación web, existen varias alternativas.

Uso del operador @

El operador @ se pone delante de una operación que puede provocar error, de manera que, si lo provoca, el programa no diga nada, no se muestre ningún mensaje. Por ejemplo, si vamos a dividir dos variables sin tener en cuenta que el divisor pueda ser cero, podríamos ponerlo así:

```
$división = @($num1/$num2);
```

Modificación del archivo php.ini

Para no mostrar los mensajes de error por pantalla, podemos editar el archivo de configuración de PHP (*php.ini*), y modificar el parámetro de configuración **display_errors**. Deberemos ponerlo *false, off o "0"* (dependiendo de la versión de PHP que tengamos), para desactivar los mensajes de error. Pero deberemos tener en cuenta que, si hacemos esto, se desactivarán para **todas las aplicaciones PHP** que tengamos en el servidor.

Uso de la instrucción die

Una tercera forma de controlar los errores que se producen en una página PHP es utilizar una instrucción **die** o **exit** en el caso de que se produzca un error en una instrucción. Por ejemplo, el siguiente código intenta ver si un fichero existe, y si no, muestra el mensaje de error indicado en la instrucción *die/exit*:

```
file_exists("fichero.txt") or die ("No se encuentra el fichero");
```

Además, con la función *die* o *exit* deja de ejecutarse el resto de la página, con lo que termina ahí su carga. Se utiliza para instrucciones que sean de vital importancia para el funcionamiento del resto del código, y normalmente al principio de la página para no dejarla con la mitad de contenido. De lo contrario, es mejor utilizar el operador @ para ocultar errores menos importantes.

Uso de excepciones

Las excepciones son un mecanismo que podemos emplear para, en el momento en que se produzca un error en alguna parte del código, se capture automáticamente y se pueda tratar en un bloque de código aparte.

Para trabajar con excepciones, debemos colocar todo el código que puede provocar error en un bloque llamado **try**. Si algo falla, automáticamente se salta a un bloque contiguo

llamado **catch**, donde podremos tratar el error y sacar el mensaje oportuno. El siguiente ejemplo trata de abrir un fichero, leerlo y guardar el contenido en un array.

```
try
{
      $contenido = file("fich1.txt");
} catch (Exception $e) {
      echo 'Se ha producido un error: ' . $e->getMessage();
}
```

Si se produce cualquier tipo de error (por ejemplo, que no exista el fichero de lectura, o no tengamos permisos para acceder a él), se provoca un error y se va directamente al bloque *catch*, mostrando el mensaje de error. Podemos utilizar la variable de tipo *Exception* que tiene el catch como parámetro, y su método *getMessage()* para mostrar el error concreto que se ha producido.

También podemos usar la instrucción **throw new Exception($mensaje)** para provocar una excepción en el caso de que alguna comprobación que hagamos nos dé un resultado incorrecto. Así provocamos un salto al "catch", o un mensaje de error en la web si no lo hacemos dentro de un *try*. Se utiliza también para algunas funciones que no provocan excepciones por sí mismas (como por ejemplo, *file_get_contents*), para provocar el error nosotros de antemano con alguna comprobación previa.

```
try
{
      if (!file_exists("fich1.txt"))
      {
            throw new Exception("El fichero de entrada no existe");
      }
      $contenido = file_get_contents("fich1.txt");
      file_put_contents("fich2.txt", $contenido);
} catch (Exception $e) {
      echo 'Se ha producido un error: ' . $e->getMessage();
}
```

Ejercicio 17

Haz tres copias del ejercicio 5.15, llamadas **copia_seguridad_arroba.php, copia_seguridad_die.php** y **copia_seguridad_excepciones.php**. En todas ellas, cambia el nombre del fichero *datos.txt* por uno que no exista en la carpeta (por ejemplo, *aa.txt*), y controla el error que puede producirse en cada caso usando una arroba, la instrucción *die* o las excepciones, para ver cómo se controla en cada caso el error y qué hace la página.

Más sobre variables

Variables globales

Veremos más adelante que existen unas variables predefinidas en PHP para ciertas tareas. Por ejemplo, la variable $_REQUEST nos permitirá acceder a los datos que el usuario nos envía desde un formulario.

Otras variables que podamos crear nosotros, tendrán su ámbito según la zona donde las creemos. Por ejemplo, si creamos una variable dentro de una función, esa variable no existirá fuera de la misma, y no podremos utilizarla. Sin embargo, si creamos una variable fuera de una función, e intentamos acceder a ella dentro, podría parecer que sí es una variable global o externa a la función, pero no es así. Veamos este ejemplo:

```
$numero = 10;
...
function incrementaNumero()
{
      $numero = $numero + 1;
      echo $numero;
}
...
incrementaNumero();
```

Si ejecutamos un código como este, el comando *echo $numero* nos dirá que número vale 1, cuando todo parece indicar que debería valer 11. La razón es que la variable externa *$numero* no es la misma que la variable interna *$numero* de la función. Esta última, al no tener un valor inicial asignado, empieza por 0, y al incrementarse vale 1, pero la variable *$numero* externa sigue valiendo 10, nadie la ha modificado.

Si queremos poder acceder a una variable externa a una función desde dentro de una función, deberemos definir en la función que esa variable es **global**, de la siguiente forma:

```
$numero = 10;
...
function incrementaNumero()
{
      global $numero;
      $numero = $numero + 1;
      echo $numero;
}
...
incrementaNumero();
```

Ahora, si ejecutamos el código, sí obtendremos lo esperado: que *$numero* vale 11.

Variables variables

Una funcionalidad muy característica de PHP y que no todos los lenguajes tienen es la posibilidad de definir variables cuyo nombre es a su vez variable. Así, una variable puede tomar su nombre según el valor de otra variable. Por ejemplo:

```
$var1 = "uno";
$$var1 = "otro uno";
```

La variable *$var1* es una variable normal, y guarda el valor "uno". Sin embargo, a la variable *$$var1* la llamamos *$$var1* en el código, pero realmente tomará su nombre según lo que valga *$var1* (en este caso, se llamaría *$uno*). Para utilizar estas variables en instrucciones tipo *echo*, dentro de comillas dobles, se pone el nombre dinámico (empezando por el segundo dólar) entre llaves:

```
echo "La segunda variable vale ${$var1}";
```

¿Qué utilidades puede tener esto? Puede parecer que es un mecanismo un tanto enrevesado sin demasiada utilidad pero, por ejemplo, puede ser bastante útil para hacer **páginas multi-idioma**. Así, nos guardaremos en diferentes variables el texto a mostrar en cada idioma, y haremos que se imprima una de ellas en función de algún parámetro adicional o variable.

```
<?php
   $texto_es = "Bienvenido";
   $texto_en = "Welcome";
   $idioma = "es";
   $texto = "texto_" . $idioma;
   echo $$texto;
?>
```

Ejercicio 18

Crea una página en la carpeta de ejercicios llamada **curriculum.php** donde, utilizando variables variables, muestres parte de tu currículum (por ejemplo, un párrafo con tus estudios y otro con los idiomas que hablas), tanto en español como en otro idioma que elijas.

Recoger datos de formulario

El medio principal que tiene el cliente (navegador) para poder enviar información al servidor son los formularios, o mediante enlaces que indican la carga de una determinada página. De cualquiera de las dos formas, se genera una **petición** al servidor, utilizando el protocolo HTTP, en la que se envían los llamados **datos de formulario** (información adicional, que normalmente son los datos que se rellenan en el formulario, de ahí su nombre).

Las variables $_GET, $_POST y $_REQUEST

Para recoger los datos que envían los clientes, tenemos predefinidas diferentes variables en PHP. Por ejemplo, si los datos se envían por método GET (cuando vienen de un enlace, o de un formulario con *method="get"*), podemos recogerlos en una variable llamada **$_GET**. Por el contrario, si se envían por método POST (para formularios con *method="post"*), podremos obtenerlos con la variable **$_POST**. Y tenemos una tercera variable, llamada **$_REQUEST**, que nos servirá para tomar los datos de cualquiera de estos dos métodos (y de otros más que no veremos aquí). Así, lo más habitual será utilizar esta tercera variable, a no ser que queramos restringir la recepción de ciertos datos sólo a métodos GET o POST.

Cualquiera de estas tres variables es un array asociativo, es decir, un conjunto de datos enviados. Para acceder a un dato concreto de ese conjunto, en general, usaremos los corchetes y dentro, entre comillas, el mismo nombre que hayamos puesto en el atributo *name* del formulario que lo envió. Por ejemplo, si tenemos un formulario como este:

```
<form action="mipagina.php" method="post">
      <label for="login">Login:</label>
      <input type="text" name="login" size="20" /><br />
      <label for="email">E-mail:</label>
      <input type="text" name="email" size="50" /><br />
      <input type="submit" value="Enviar" />
</form>
```

al enviarlo, en la página *mipagina.php* podríamos acceder al login que haya puesto el usuario cliente mediante:

```
$loginUsuario = $_REQUEST["login"];
```

o también así:

```
$loginUsuario = $_POST["login"];
```

También podemos, antes de acceder a algún dato y dar por supuesto que se ha enviado, comprobar con *isset* si realmente ese dato existe dentro de la variable $_REQUEST (si no existe es porque no se ha enviado dicho dato):

```
if (isset($_REQUEST['login']))
      ...
```

Este mecanismo de obtener datos nos servirá para casi todos los campos de formularios, salvo aquellos que puedan devolver múltiples valores, como las listas con selección múltiple, caso que veremos a continuación.

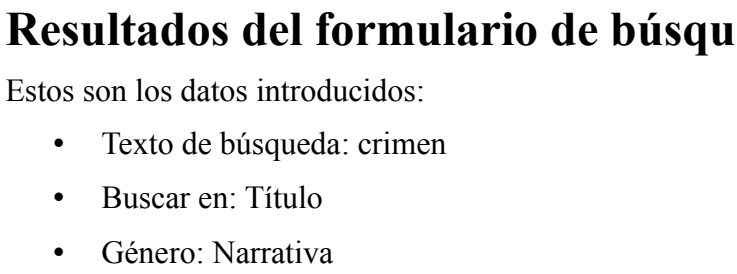
Ejercicio 19

Crea una subcarpeta llamada **libros** en la carpeta de ejercicios. Dentro, crea dos páginas PHP:

- Una página llamada **form_libros.php** con un formulario como el siguiente:

  El formulario debe enviarse a una página llamada *result_libros.php* que haremos a continuación. En la lista desplegable deben aparecer los géneros *Narrativa, Libros de texto* y *Guías y mapas*.

- Una página llamada **result_libros.php** que muestre un resumen de lo que se ha enviado por el formulario, y tenga un enlace para volver al formulario anterior:



Ejercicio 20

Crea una subcarpeta llamada **departamentos** en tu carpeta de ejercicios. Dentro, crea estas dos páginas PHP:

- Una página llamada **form_dep.php** con un formulario que mostrará una lista desplegable con 4 nombres de departamento: INFORMÁTICA, LENGUA, MATEMÁTICAS e INGLÉS. El usuario podrá elegir el departamento al que pertenece, y pulsar el botón de Enviar. El formulario se enviará a la página *presupuesto.php* que haremos a continuación
- La página **presupuesto.php** deberá recoger el departamento indicado por el usuario y, dependiendo de cuál sea, le indicará el presupuesto asignado a ese departamento, según la siguiente lista:
  - INFORMÁTICA: 500 euros
  - LENGUA: 300 euros
  - MATEMÁTICAS: 300 euros
  - INGLÉS: 400 euros

Recogida de campos múltiples

En el caso de campos con múltiples valores enviados (por ejemplo, una lista de selección múltiple), el campo del formulario tendrá este aspecto:

```
<form...>
	...
	<select multiple name="personas[]" size="5">
		<option value="Juan Rodríguez">Juan Rodríguez</option>
		...
	</select>
...
```

Y al enviarse, recogeremos los elementos enviados en una variable que podremos recorrer con un *foreach* o una estructura similar:

```
<?php
	$personas = $_REQUEST['personas'];
	foreach ($personas as $persona)
	{
		...
	}
...
```

Envío de datos mediante enlaces

Hemos dicho que a través de los enlaces también podemos enviar datos al servidor. Normalmente, los enlaces no envían nada más que la página o recurso que queremos ver (por ejemplo, *http://www.google.es*). Pero también podemos añadir, al final de la URL, nombres de parámetros y sus respectivos valores, separados por el símbolo &, como si los enviáramos desde un formulario. Así, si quisiéramos "simular" el envío del formulario anterior para un usuario con login "usu1" y e-mail "usu1@gmail.com", pondríamos un enlace así:

```
<a href="mipagina.php?login=usu1&email=usu1@gmail.com">Enviar datos</a>
```

Estos datos podremos recogerlos a través de las variables $_GET o $_REQUEST (el método *POST* no se emplea en los enlaces):

```
$mailUsuario = $_GET["email"];
```

Ejercicio 21

Modifica el ejercicio anterior, creando una página más llamada **form_dep2.php** que, en lugar de un formulario con la lista desplegable, muestre al usuario 4 enlaces con los 4 nombres de departamento. Los enlaces deben enviar (todos) a la página *presupuesto.php*, indicando cuál es el departamento seleccionado, para que obtengamos el mismo resultado que con el formulario anterior.

Envíos a la propia página

Es habitual también encontrarnos con formularios cuyo *action* apunta a la misma página del formulario. Después, con código PHP (con instrucciones *if..else*), podemos distinguir si queremos cargar una página normal, o si hemos recibido datos del formulario y hay que procesarlos.

En cualquier caso, para poder hacer que un formulario se envíe a su propia página, podemos directamente poner el nombre de la misma página en el action (por ejemplo, *action="mipagina.php"*), pero si más adelante decidimos cambiar el nombre del archivo, corremos el riesgo de olvidarnos cambiar el formulario también. Para evitar este problema, contamos en PHP con una variable llamada **$_SERVER['PHP_SELF']**, con lo que bastaría con poner esa variable en el *action* del formulario :

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
...
</form>
```

Ejercicio 22

En el ejercicio anterior, crea una página llamada **presupuesto2.php** que tenga el formulario con la lista desplegable de departamentos, y se llame a sí misma para calcular el presupuesto según lo que se envíe por su propio formulario.

Subir ficheros al servidor

Otra forma de comunicación entre cliente y servidor es la subida de archivos del cliente al servidor como parte de un formulario. Para ello, añadimos al formulario un campo *input* de tipo *file*. Además, debemos añadir a la etiqueta *form* el atributo *enctype*, con el valor *multipart/form-data* para hacer saber al navegador y al servidor que en ese formulario se van a enviar los datos de un archivo.

```
<form action="pagina.php" method="post" enctype="multipart/form-data">
      ...
      <input type="file" name="fichero" />
```

El tamaño máximo permitido del fichero a subir se puede establecer en el archivo *php.ini* (propiedad *upload_max_filesize*), o añadiendo un campo oculto (*type="hidden"*) en el formulario, con nombre *MAX_FILE_SIZE* y el máximo tamaño permitido, en bytes.

```
<form action="pagina.php" method="post" enctype="multipart/form-data">
      <input type="hidden" name="MAX_FILE_SIZE" value="20000" />
      <input type="file" name="fichero" />
```

Una vez recibido el formulario por el servidor, el fichero se almacena en una carpeta temporal, y debemos moverlo a su ubicación definitiva con la función **move_uploaded_file**. La variable global **$_FILES** contiene toda la información de los archivos que se han subido con el formulario. Un típico código de subida podría ser:

```
if (is_uploaded_file ($_FILES['fichero']['tmp_name']))
{
   $nombreDirectorio = "ficheros/";
   $idUnico = time();
   $nombreFichero = $idUnico . "-" . $_FILES['fichero']['name'];
   move_uploaded_file ($_FILES['fichero']['tmp_name'], $nombreDirectorio .
                       $nombreFichero);
} else
   print ("No se ha podido subir el fichero\n");
```

Además de la clave *name*, la variable *$_FILES* también permite localizar otra información del fichero subido, como el tipo de archivo (clave *type*), o el tamaño (clave *size*). En el código, la variable *$idUnico* se utiliza para no sobreescribir archivos con el mismo nombre, sino ponerles una marca temporal como prefijo para almacenar un histórico. Pero podría evitarse la variable y sobreescribir archivos que se llamen igual:

```
if (is_uploaded_file ($_FILES['fichero']['tmp_name']))
{
   $nombreDirectorio = "ficheros/";
   move_uploaded_file ($_FILES['fichero']['tmp_name'], $nombreDirectorio .
                       $_FILES['fichero']['name']);
} else
   print ("No se ha podido subir el fichero\n");
```

Ejercicio 23

Crea en tu carpeta de ejercicios una subcarpeta llamada *ficheros*. Dentro, crea un formulario llamado **form_imagen.php** que permita seleccionar un archivo y subirlo a una página llamada **subir_imagen.php**. En esta página, comprobaremos que el fichero subido es una imagen, y si lo es, lo copiaremos a una subcarpeta *imgs* y mostraremos la imagen subida en la misma página *subir_imagen.php*.

Redirecciones e inclusiones

PHP también dispone de instrucciones para permitir incluir el contenido de un documento en otro como parte de la respuesta a un cliente, o redirigir a otra página si es necesario.

Redirigir a otra página

Las redirecciones se producen cuando desde el cliente accedemos a una página que está preparada para, según ciertas condiciones, enviarnos a otra. Por ejemplo, si intentamos acceder a una zona restringida y no estamos logueados en el servidor, lo normal es que se nos redirija a la página de login para que entremos con nuestro usuario y contraseña.

La redirección de una página a otra se hace aprovechando el protocolo HTTP, mediante una de sus cabeceras, llamada ***Location***. Usaremos el comando **header** para acceder a las cabeceras HTTP, y dentro pondremos la cabecera *Location*, junto con la página a la que queremos redirigir. Por ejemplo, si queremos redirigir a la página *login.php* de nuestra web, pondríamos algo como:

```
header("Location:login.php");
```

Es importante recalcar que las redirecciones deben hacerse **antes** de escribir cualquier contenido por el buffer de salida, es decir, antes de generar cualquier tipo de contenido HTML que enviar al navegador.

También podemos añadir algo más de información con la redirección. Por ejemplo, redirigir pasados X segundos, para así mostrar algún mensaje de advertencia en la página, y que el usuario sepa que va a ser redirigido. Para ello se utiliza la cabecera ***Refresh***, indicando el tiempo en segundos a esperar, y un atributo *url* con la página a la que redirigir (si no se pone esto, se recarga la misma página pasado ese tiempo, lo que puede ser útil, por ejemplo, para actualizar resultados deportivos). El siguiente código redirige a *index.php* pasados 5 segundos, mostrando un mensaje de redirección.

```
header("Refresh:5; url=index.php");
...
echo '<p>En breve le redirigiremos a la página principal.</p>';
```

Ejercicio 24

Modifica la página *result_libros.php* de ejercicios anteriores para que, si el texto a buscar que le llega del formulario está vacío, o no le han llegado datos del formulario (por ejemplo, porque hemos cargado esta página directamente, sin pasar por el formulario), redirija a la página *form_libros.php* para rellenar los datos. Opcionalmente, trata de mostrar un mensaje de error en color rojo, en el formulario, indicando que "Debes rellenar todos los campos antes de enviar el formulario".

Incluir páginas dentro de otras

La inclusión de contenido externo es una funcionalidad importante para, por ejemplo y sobre todo, no tener que estar repitiendo código en diferentes páginas. Así, podríamos tener una página *cabecera.php* con el encabezado de nuestra página web (bloque *head*, logo de la empresa y menú de navegación principal):

```
<html>
      <head>
            ...
      </head>
      <body>
            <img class="logo" src="imgs/logoempresa.jpg" />
            <ul class="menuNavegacion">
                  <li><a href="inicio.php">Inicio</a></li>
                  <li><a href="noticias.php">Noticias</a></li>
                  ...
            </ul>
```

Después, podríamos incluir este fichero *cabecera.php* en cada página de nuestra web, para así tener la misma cabecera en todas, y modificar simplemente el contenido de la página. Para ello, podemos utilizar las instrucciones o directivas **include, include_once, require** o **require_once**. Vamos a ver el ejemplo con *include*, y después comentaremos qué diferencias hay entre estas cuatro directivas:

```
<?php include('cabecera.php'); ?>

            <div class="principal">
                  ...
            </div>
            ...
      </body>
</html>
```

Cuando incluimos un documento en otro, el funcionamiento es como si copiáramos y pegáramos literalmente todo el código fuente del fichero incluido en el fichero destino. Pero tenemos la ventaja de que, de este modo, si queremos hacer algún cambio (por ejemplo, añadir más opciones al menú de la cabecera), se cambiaría el fichero *cabecera.php* y automáticamente se actualizarían los demás donde se incluye.

Como hemos dicho, podemos incluir ficheros utilizando cuatro directivas: *include, include_once, require* o *require_once*. Las diferencias entre ellas son las siguientes:

- Las directivas *include* e *include_once* provocan una advertencia si no se encuentra el archivo a incluir, mientras que *require* y *require_once* producen un error fatal que cuelga la página (son más restrictivas estas últimas)
- Las directivas acabadas en *_once* incluyen el documento si no se ha incluido ya previamente con otra directiva *include* o *require.* Esto es habitual cuando incluimos ficheros que, a su vez, incluyen otros, y otros... Es posible que en estos casos alguna de las inclusiones afecte a un fichero que pensábamos incluir, y ya está incluido.

Lógicamente, de la misma forma que hemos incluido la cabecera, también podríamos tener otro fichero *pie.php* con el pie de página (cierre de etiquetas, marco legal, copyright, etc.), e incluirlo también en todas las páginas.

Por último destacar que, a la hora de incluir páginas dentro de otras, se suele utilizar la siguiente convención:

- Si la página incluida no es una página completa por sí misma (sino que necesita de otras para completarse), o es una librería de funciones PHP para poder utilizar en la página donde se incluye, el archivo suele tener extensión **.inc**. Así, por ejemplo, la cabecera anterior no sería una página completa por sí misma, y debería haberse

llamado *cabecera.inc*. De la misma forma, si hacemos un conjunto de funciones para acceder a una base de datos, y lo queremos incluir en un archivo PHP para usar esas funciones, el archivo debería llamarse, por ejemplo, *libreria.inc*.

- Si la página incluida es un archivo completo por sí mismo, utilizaremos la extensión habitual *.php*.

Ejercicio 25

Modifica el ejercicio de *libros* anterior. Añade en la carpeta un nuevo archivo llamado **cabecera.inc**, con la cabecera de las dos páginas web (el *html, head*, algún estilo CSS como color de fondo o tipo de letra, y algún logo en el *body* que encuentres por Internet relacionado con el mundo de los libros). Después, haz que las otras dos páginas (*form_libros.php* y *result_libros.php*) incluyan la nueva cabecera y quiten las suyas propias, para compartir cabecera.

Cookies y sesiones

Veremos ahora dos mecanismos para intercambiar variables entre cliente y servidor: las cookies y las sesiones.

Cookies

Las cookies son básicamente un texto que se intercambian navegador y servidor entre distintas peticiones, con información acerca de las mismas y de lo que se ha estado haciendo. Al cerrar el navegador o la conexión, éste se guarda en ficheros locales dicha información, para poder seguir compartiéndola en futuras visitas. Sin embargo, las cookies tienen una fecha de caducidad preestablecida (dependiendo de la cookie), y pasado ese tiempo (salvo que se renueve con una nueva visita a la página), caducan y desaparecen.

La principal utilidad de las cookies es recordar a los usuarios que acceden a una web, y las cosas que hacen, pero esta capacidad también sirve para que otros rastreen esa actividad y puedan interceptar información delicada (como usuarios y contraseñas). Por ello, actualmente los navegadores permiten desactivar y/o borrar las cookies, y los sitios web que las utilizan están obligados a avisar a los usuarios de ello. Como programadores web, las cookies deben usarse para almacenar información no relevante del usuario (preferencias del sitio, lugares favoritos o, como mucho, el login para que no lo tenga que volver a escribir, pero no el password).

Hay que tener en cuenta, además, ciertos aspectos relacionados con las cookies:

- Son independientes para cada servidor, es decir, el navegador almacena y gestiona de forma separada las cookies de cada sitio web que visitamos, y unas no tienen nada que ver con otras.
- No se pueden transmitir virus, ni acceder al disco duro, desde las cookies. Es el navegador quien controla sus propios ficheros de cookies y quien los gestiona.

Ahora que ya sabemos un poco más sobre las cookies, veamos cómo crearlas y mantenerlas en PHP. Para crear una cookie se empleará la función **setcookie**, a la que le pasaremos como parámetros el nombre de la cookie, su valor, y el tiempo en que expirará, desde ahora. Este último dato suele utilizar la función *time()* para contar el tiempo desde ahora, y añadirle los segundos que queramos. Si ponemos 0 o no ponemos nada, durará hasta que el usuario termine su sesión (cierre sesión, o cierre el navegador). Si ponemos un valor negativo o pasado, la cookie se borrará. Esto se usa para borrar a mano las cookies y no esperar a que lo haga el navegador.

Veamos un ejemplo donde creamos una cookie llamada *colorFavorito* con un color hexadecimal que luego podríamos usar, por ejemplo, para establecer ese color de fondo para el usuario. Asignaremos a la cookie una vida de 1 hora (3600 segundos):

```
setcookie("colorFavorito", "#432FA1", time()+3600);
```

Si más adelante queremos consultar esta cookie (para extraer el color y poderlo poner como estilo, por ejemplo), usaremos el array predefinido **$_COOKIE**, pasándole entre corchetes el nombre de la cookie, y obtendremos el valor asociado a la misma.

```
$color = $_COOKIE["colorFavorito"];
echo "<style>body { background-color: " . $color . "; }</style>";
```

Ejercicio 26

Crea una carpeta llamada **preferencias** en la carpeta de trabajo del tema. Dentro, crea las siguientes páginas PHP:

- Página **preferencias.php** con un formulario que le pida al usuario su nombre y su color favorito (a elegir de una lista desplegable, o con el control *input type="color"* de HTML5). El formulario deberá enviarse a *guarda_prefs.php*
- Página **guarda_prefs.php** que recogerá los datos del formulario anterior, y los guardará en dos cookies llamadas *nombreusu* y *colorusu*, con 5 minutos de duración. Esta página redirigirá (con *header*) a la página *index.php* una vez almacenadas las cookies.
- Página **index.php** que, si hay cookies guardadas del usuario, mostrará una página con el color de fondo preferido y un mensaje personalizado de bienvenida (por ejemplo, "Bienvenido, Nacho"). Si no hay cookies guardadas, mostrará una página con fondo blanco y el mensaje "Página de inicio". En cualquiera de los dos casos, habrá un enlace para ir al formulario *preferencias.php*.

Opcionalmente, añade una página **borrar_prefs.php** que elimine las preferencias (cookies) del usuario actual, y redirija a *index.php*. Pon un enlace *"Borrar preferencias"* en el *index.php* que llame a esta página, pero que se muestre sólo cuando el usuario actual tenga preferencias guardadas.

Sesiones

Las sesiones son otro mecanismo de intercambio de información entre navegador y servidor, cuyo tiempo de vida es igual a toda la visita de un usuario a una web (en cuanto el usuario cierra sesión, desconecta o cierra el navegador, se pierde la información). Al igual que ocurre con las cookies, las sesiones que tengamos en dos sitios web diferentes son independientes entre sí, al igual que las sesiones que tengan dos usuarios diferentes en un mismo sitio web.

Las principales diferencias entre las sesiones y las cookies son que las sesiones no se almacenan en ficheros en el disco, podemos almacenar en ellas cualquier información (no sólo textos), y no se pueden desactivar desde el navegador.

Para manejar datos en una sesión, primero deberemos crearla con la función **session_start()**. Esta función crea la sesión si no estaba creada, y en caso de que ya exista, utiliza esa sesión. Es aconsejable poner esta función al principio de todas las páginas que necesiten sesiones.

Después, utilizaremos el array **$_SESSION** para guardar y recuperar datos de la sesión. Por ejemplo, si queremos guardar el login del usuario que nos ha enviado por un formulario, pondríamos algo como:

```
$_SESSION["loginUsuario"] = $_REQUEST["login"];
```

Si queremos eliminar algún dato de la sesión, usaremos la función **unset** con dicho dato del array:

```
unset($_SESSION["loginUsuario"]);
```

Si queremos cerrar la sesión entera (y perder todo lo que haya en el array *$_SESSION*), usamos la función **session_destroy()**. Es aconsejable borrar antes a mano todas las variables que hayamos creado en la sesión, para que no se queden ocupando memoria. Para ello, en lugar de hacer *unset* con cada variable, también podemos hacer:

```
$_SESSION = array();
```

Ejercicio 27

Crea una página llamada **carro.php** con una lista de enlaces de diferentes artículos y su precio. Por ejemplo:

- Zapatillas Reebok (40 euros)
- Sudadera Domyos (15 euros)
- Pala de pádel Vairo (50 euros)
- Pelota de baloncesto Molten (20 euros)

Cada vez que el usuario haga clic en un artículo, se enviará a la propia página, recogerá el precio del artículo (que se le enviará como parámetro) y acumulará en una variable llamada *carro* en sesión cuánto llevamos comprado hasta el momento. Por ejemplo:

Total comprado: 55 euros

- Zapatillas Reebok (40 euros)
- ...

Añade un enlace a la página que sea "Vaciar carro" que limpie el carro y deje así el total a 0.

Ejercicio 28

Crea una carpeta llamada **login** en la carpeta de ejercicios. Dentro vamos a crear un pequeño sistema de login y uso de sesiones. Crea los siguientes archivos:

- Un archivo **usuarios.txt** donde vamos a guardar varios logins y passwords de prueba, separados por ":", uno por fila, como en este ejemplo:

  usu1:pass1
  usu2:pass2
  prueba:passprueba

- Después, crea una página **cabecera.inc** que hará lo siguiente:
  - Comprobará si existe en sesión un atributo llamado *loginusu*. Si no existe, redirigirá a la página *login.php* que haremos después. Si existe, sacará un menú de enlaces a las páginas *pag1.php* y *pag2.php*
  - Crea las páginas **index.php**, **pag1.php** y **pag2.php** que incluyan la cabecera anterior (con *require*), y como contenido, ponles simplemente un mensaje de bienvenida diferente para cada página (para identificarlas).
  - Finalmente, la página de **login.php** tendrá un formulario para pedirle al usuario su login y password, y se llamará a sí misma. Los datos que reciba del formulario, los comparará con los del fichero *usuario.txt* para ver si coincide con alguno. Si coincide, creará un atributo *loginusu* en sesión con el login del

usuario y redirigirá a *index.php*. Si no coincide con ninguno, se volverá a cargar el propio formulario, con un mensaje de error en rojo indicando que el login o la contraseña son incorrectos.

Cuándo usar cookies o sesiones

Ya hemos visto estos dos mecanismos de compartir y actualizar información entre cliente y servidor, pero... ¿cuándo es conveniente utilizar cookies y cuándo sesiones? Básicamente debemos regirnos por estas sencillas reglas:

- Usaremos **cookies** para almacenar información simple (texto) y no relevante para el funcionamiento de la web (porque los navegadores pueden desactivar las cookies). Por ejemplo, las últimas búsquedas que ha hecho un usuario en una web, o el login del usuario para recordárselo en el formulario la próxima vez que intente loguearse (aunque el login también deberemos almacenarlo en sesiones para recordar quién ha entrado hasta que cierre sesión).
- Usaremos **sesiones** cuando queramos almacenar datos complejos, o cruciales para el funcionamiento de la web, ya que no pueden ser desactivadas por el navegador. Por ejemplo, el login del usuario (no el password) es útil almacenarlo en sesiones, para tener la certeza de poder identificarlo en cada página que visita (si estuviera en cookies, podrían desactivarse y perder quién ha entrado). Los elementos de un carro de la compra que tengamos añadidos de una tienda virtual, también se deberían almacenar en sesiones, pues son (o pueden ser) datos complejos, no simple texto.
- No usaremos ni una ni otra para almacenar información privada del usuario, como por ejemplo contraseña, número de tarjeta de crédito, etc.