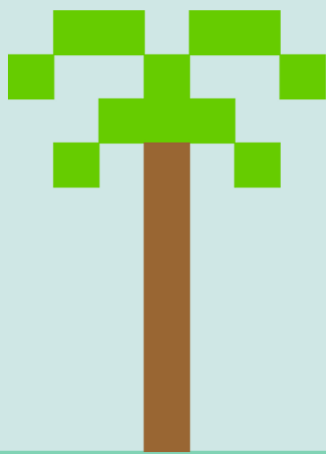


{ LUIS JOSÉ SÁNCHEZ }

APRENDE
PHP
CON EJERCICIOS



[INCLUYE MÁS DE 100 EJERCICIOS RESUELTOS]

Aprende PHP con Ejercicios

Luis José Sánchez González

Este libro está a la venta en <http://leanpub.com/aprendephpconejercicios>

Esta versión se publicó en 2016-01-19



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2014 - 2016 Luis José Sánchez González

Índice general

Sobre este libro	i
Sobre el autor	ii
Introducción	iii
1. Conceptos básicos: Integración de PHP en HTML. Variables. Operadores	1
1.1 Integración de PHP en HTML ¡Hola mundo!	1
1.2 Variables	2
Definición de variables	2
Operadores aritméticos	3
1.3 Ejercicios	5
2. Recogida de datos por teclado mediante formularios	7
2.1 Recogida de datos en dos pasos	7
2.2 Métodos get y post	9
2.3 Ejercicios	10
3. Sentencia condicional (if y switch)	12
3.1 Sentencia if	12
3.2 Operadores de comparación	13
3.3 Operadores lógicos	14
3.4 Sentencia switch	15
3.5 Carga reiterada de una página. Adivina el número.	17
3.6 Ejercicios	19
4. Bucles	22
4.1 Bucle for	22
4.2 Bucle while	22
4.3 Bucle do-while	23
4.4 Ejercicios	25
5. Arrays	29
5.1 Arrays clásicas	29
5.2 Arrays asociativos	31
5.3 Arrays bidimensionales	32
5.4 Iterador foreach	33

ÍNDICE GENERAL

5.5	Cómo recoger datos para un array mediante un formulario	34
5.6	Ejercicios	36
6.	Funciones	40
6.1	Implementando funciones para reutilizar código	40
6.2	Creación de bibliotecas de funciones	42
6.3	¿Se pueden sobrecargar las funciones en PHP?	44
6.4	Ejercicios	47
7.	Sesiones y cookies	49
7.1	Sesiones	49
7.2	Cookies	51
7.3	Ejercicios	54
8.	Acceso a bases de datos	57
8.1	Acceso a BBDD desde PHP	57
8.2	Extensión mysql	57
	Establecimiento de una conexión	57
	Listado completo de una tabla	59
	Búsqueda de un valor en una tabla	60
	Tratamiento de errores	61
8.3	Extensión mysqli	62
	Establecimiento de una conexión	62
	Listado completo de una tabla	63
8.4	PDO (<i>PHP Data Objects</i>)	65
	Establecimiento de una conexión	65
	Listado completo de una tabla	65
8.5	Operaciones sobre una tabla	67
8.6	Ejercicios	69
9.	PHP orientado a objetos	71
9.1	El paradigma de la Programación Orientada a Objetos	71
9.2	Encapsulamiento y ocultación	71
9.3	Implementación de clases en PHP	72
9.4	Herencia	78
9.5	Atributos y métodos de clase (<i>static</i>)	82
9.6	Serialización de objetos	84
9.7	Ejercicios	87
10.	Modelo Vista Controlador	88
10.1	El modelo	88
10.2	La vista	91
10.3	El controlador	93
10.4	Ejercicios	95
11.	Twig - Motor de plantillas	96
11.1	Introducción	96

ÍNDICE GENERAL

11.2	“Hola mundo” con Twig	96
11.3	Paso de información a las plantillas	97
11.4	Uso de Twig en el Modelo Vista Controlador	102
11.5	Herencia de plantillas	104
11.6	Ejercicios	109

Sobre este libro

“Aprende PHP con Ejercicios” es un manual práctico para aprender a programar en lenguaje PHP desde cero.

No es necesario tener conocimientos previos aunque el lector que ya conozca algún lenguaje de programación avanzará más rápido en la asimilación de los contenidos. La dificultad del libro es gradual, empieza con conceptos muy básicos y ejercicios muy sencillos y va aumentando en complejidad y dificultad a medida que avanzan los capítulos.

Este libro contiene más de 100 ejercicios. Tanto las soluciones a los ejercicios como los ejemplos están disponibles en GitHub: <https://github.com/LuisJoseSanchez/aprende-php-con-ejercicios>.

“Aprende PHP con Ejercicios” es un libro hecho casi a medida de la asignatura *“Desarrollo Web en Entorno Servidor”* que forma parte del currículo del segundo curso del ciclo formativo DAW (Desarrollo de Aplicaciones Web) pero igualmente puede ser utilizado por estudiantes de cualquier curso relacionado con el desarrollo web en el que se utilice PHP.

Sobre el autor

Luis José Sánchez González es Ingeniero Técnico en Informática de Gestión por la Universidad de Málaga (España) y funcionario de carrera de la Junta de Andalucía desde 1998. En su trabajo de profesor de Informática combina sus dos pasiones: la enseñanza y la programación.

En el momento de publicar este libro, es profesor del I.E.S. Campanillas (Málaga) e imparte clases en el Ciclo Superior de Desarrollo de Aplicaciones Web.

Para cualquier duda, comentario o sugerencia en relación a este libro, puedes ponerte en contacto con el autor mediante la dirección de correo electrónico luisjoseprofe@gmail.com

Introducción

PHP (Personal Home Page) es, según el [índice PYPL](https://sites.google.com/site/pydatalog/pypl/PyPL-PopularitY-of-Programming-Language)¹ (PopularitY of Programming Language index), el segundo lenguaje de programación más utilizado en el mundo, únicamente superado por Java².

PHP es un lenguaje de programación estructurado y, como tal, hace uso de variables, sentencias condicionales, bucles, funciones... PHP es también un lenguaje de programación orientado a objetos y, por consiguiente, permite definir clases con sus métodos correspondientes y crear instancias de esas clases.

Es muy frecuente combinar PHP con HTML y Javascript por lo que, para sacar provecho del libro, es recomendable tener unos conocimientos básicos sobre estas materias. Una web excelente para aprender HTML y Javascript es [W3Schools.com](http://w3schools.com)³

A diferencia de JavaScript o HTML que se ejecutan en el navegador, PHP se ejecuta en el servidor, por eso nos permite acceder a los recursos que tiene el servidor, por ejemplo a una base de datos. Los programas escritos en PHP se ejecutan en el servidor y el resultado se envía al navegador. El resultado es normalmente una página HTML.

Al ser PHP un lenguaje que se ejecuta en el servidor no es necesario que el navegador lo soporte, es independiente del navegador, pero sin embargo para que las páginas PHP funcionen, el servidor donde están alojadas debe soportar PHP. Hoy en día la práctica totalidad de servidores que ofrecen servicios de *hosting* soportan PHP por defecto.

Los programas necesarios para probar los ejemplos de este libro y para realizar los ejercicios son los siguientes:

1. Un navegador web
2. Un editor de texto
3. El lenguaje PHP
4. Un servidor web (p. ej. Apache)
5. El módulo de PHP para el servidor
6. Un gestor de bases de datos como por ejemplo MySQL.

Afortunadamente hay paquetes que ya incluyen un entorno con editor de texto, el lenguaje PHP y un servidor web; todo ello convenientemente configurado y preparado para empezar a programar sin tener que preocuparnos de nada que no sea nuestro programa PHP. Para la realización de los programas de este libro se ha utilizado el entorno NetBeans. La última versión se puede descargar de forma gratuita desde [la página oficial de Netbeans](http://netbeans.org)⁴.

¹<https://sites.google.com/site/pydatalog/pypl/PyPL-PopularitY-of-Programming-Language>

²Para aprender a programar en Java recomendamos el libro "Aprende Java con Ejercicios" de Luis José Sánchez.

³<http://w3schools.com>

⁴<http://netbeans.org>

1. Conceptos básicos: Integración de PHP en HTML. Variables. Operadores

1.1 Integración de PHP en HTML ¡Hola mundo!

Abre el entorno NetBeans y selecciona **Archivo** → **Proyecto Nuevo...** A continuación selecciona **PHP** en el apartado **Categorías** y **Aplicación PHP** en el apartado **Proyectos**. Dale un nombre al proyecto, por ejemplo **Saludo**. Haz clic en **Siguiente** hasta que salga la ventana del editor. Por defecto, se crea un archivo con el nombre `index.php`. Edítalo y escribe el siguiente código:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      echo "¡Hola mundo!";
    ?>
  </body>
</html>
```

Por último, dale a la tecla **F6** para ejecutar el proyecto.

Como habrás observado, el programa contiene código en HTML mezclado con una sentencia en PHP. Cada vez que quieras insertar código en PHP, deberás encerrarlo entre las etiquetas `<?php` y `?>`.

La instrucción `echo` sirve para volcar texto en la página HTML. No es necesariamente el texto que se quiere mostrar. Veamos otro ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- Utilizo PHP para poner en negrita una palabra -->
    Hola
    <?php echo "<b><u>"; ?>
    mundo
    <?php echo "</u></b>"; ?>
```

```
</body>
</html>
```

Observa que esta vez, echo ha servido para volcar en HTML las etiquetas y <u> que hacen que una cadena de caracteres se muestre en negrita y en cursiva respectivamente. Fíjate que después de una sentencia en PHP se escribe un punto y coma.

Aquí tienes otro ejemplo, en este caso mostramos una línea utilizando HTML y otra utilizando PHP:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- Muestra una frase con HTML -->
    Hola mundo<br>
    <!-- Muestra una frase con PHP -->
    <?php echo "Es muy fácil programar en PHP."; ?>
  </body>
</html>
```

1.2 Variables

Definición de variables

Una variable es un contenedor de información, es algo así como una cajita que tiene un nombre y en la que podemos meter un valor. Las variables pueden almacenar números enteros, números decimales, caracteres, cadenas de caracteres (palabras o frases), etc. El contenido de las variables se puede mostrar y se puede cambiar durante la ejecución de una página PHP (por eso se llaman variables).

Los nombres de las variables comienzan con el símbolo del dólar (\$) y no es necesario definirlos como se hace en otros lenguajes de programación como C, Java, Pascal, etc. La misma variable puede contener un número y luego el nombre de una ciudad, no existe restricción en cuanto al tipo como en la mayoría de los lenguajes.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $x = 24;
      $pi = 3.1416;
      $animal = "conejo";
```

```

    $saludo = "hola caracola";
    echo $x, "<br>", $pi, "<br>", $animal, "<br>", $saludo;
    ?>
</body>
</html>

```

En este ejemplo se han definido las variables `$x`, `$pi`, `$animal` y `$saludo`. Con la instrucción `echo` se ha mostrado el valor que contienen, insertando un salto de línea entre ellas. Fíjate que la coma sirve para unir trozos de una cadena de caracteres.

Operadores aritméticos

Los operadores de PHP son similares a los de cualquier otro lenguaje de programación. Estos son los operadores que se pueden aplicar tanto a las variables como a las constantes numéricas:

Operador	Nombre	Ejemplo	Descripción
+	suma	<code>20 + \$x</code>	suma dos números
-	resta	<code>\$a - \$b</code>	resta dos números
*	multiplicación	<code>10 * 7</code>	multiplica dos números
/	división	<code>\$altura / 2</code>	divide dos números
%	módulo	<code>5 % 2</code>	devuelve el resto de la división entera
++	incremento	<code>\$a++</code>	incrementa en 1 el valor de la variable
--	decremento	<code>\$a--</code>	decrementa en 1 el valor de la variable

A continuación se muestra un programa que ilustra el uso de estos operadores:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $a = 8;
      $b = 3;
      echo $a + $b, "<br>";
      echo $a - $b, "<br>";
      echo $a * $b, "<br>";
      echo $a / $b, "<br>";
      $a++;
      echo $a, "<br>";
      $b--;
      echo $b, "<br>";
    ?>
  </body>

```

```
</html>
```

Mientras depuramos un programa, con frecuencia necesitamos ver el valor de las variables. Puedes hacer echo sobre cada una de ellas como hemos visto en los ejemplos anteriores pero es muy cómodo usar

```
print_r(get_defined_vars());
```

que muestra el valor de todas y cada una de las variables que se han definido.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $numero = 20;
      $palabra = "hola";
      print_r(get_defined_vars());
    ?>
  </body>
</html>
```

1.3 Ejercicios



Ejercicio 1

Escribe un programa que muestre tu nombre por pantalla. Utiliza código PHP.



Ejercicio 2

Modifica el programa anterior para que muestre tu dirección y tu número de teléfono. Cada dato se debe mostrar en una línea diferente. Mezcla de alguna forma las salidas por pantalla, utilizando tanto HTML como PHP.



Ejercicio 3

Escribe un programa que muestre por pantalla 10 palabras en inglés junto a su correspondiente traducción al castellano. Las palabras deben estar distribuidas en dos columnas. Utiliza la etiqueta `<table>` de HTML.



Ejercicio 4

Escribe un programa que muestre tu horario de clase mediante una tabla. Aunque se puede hacer íntegramente en HTML (igual que los ejercicios anteriores), ve intercalando código HTML y PHP para familiarizarte con éste último.



Ejercicio 5

Escribe un programa que utilice las variables `$x` y `$y`. Asigne los valores 144 y 999 respectivamente. A continuación, muestra por pantalla el valor de cada variable, la suma, la resta, la división y la multiplicación.



Ejercicio 6

Crea la variable `$nombre` y asígnele tu nombre completo. Muestra su valor por pantalla de tal forma que el resultado sea el mismo que el del ejercicio 1.



Ejercicio 7

Crea las variables `$nombre`, `$direccion` y `$telefono` y asígneles los valores adecuados. Muestra los valores por pantalla de tal forma que el resultado sea el mismo que el del ejercicio 2.



Ejercicio 8

Realiza un conversor de euros a pesetas. La cantidad en euros que se quiere convertir deberá estar almacenada en una variable.



Ejercicio 9

Realiza un conversor de pesetas a euros. La cantidad en pesetas que se quiere convertir deberá estar almacenada en una variable.



Ejercicio 10

Escribe un programa que pinte por pantalla una pirámide rellena a base de asteriscos. La base de la pirámide debe estar formada por 9 asteriscos.



Ejercicio 11

Igual que el programa anterior, pero esta vez la pirámide estará hueca (se debe ver únicamente el contorno hecho con asteriscos).



Ejercicio 12

Igual que el programa anterior, pero esta vez la pirámide debe aparecer invertida, con el vértice hacia abajo.

2. Recogida de datos por teclado mediante formularios

2.1 Recogida de datos en dos pasos

En una página web, los datos se introducen mediante formularios. En la mayoría de las ocasiones tendremos dos páginas: una página con un formulario que recoge los datos y otra página que los procesa.

A continuación tenemos una página con nombre `index.php` (se podría llamar también `index.html` ya que no contiene código PHP):

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    Introduce tu nombre:
    <form action="saluda.php" method="get">
      <input type="text" name="nombre"><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

Al pulsar el botón **Enviar**, el contenido del formulario se envía a la página que indicamos en el atributo `action`. La página `saluda.php` tendría el siguiente código:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    Hola <?php echo $_GET['nombre'] ?>,
    que tengas un buen día.
  </body>
</html>
```

Observa que la información que se quiere enviar se introduce en un cuadro de texto dentro del formulario (página `index.php`). A esa información la llamamos en este caso `nombre`. Para recoger esa información se utiliza `$_GET['nombre']`.

Cuando los datos que se recogen y se manipulan son números en lugar de cadenas de caracteres, el procedimiento es el mismo.

En el siguiente ejemplo tenemos una aplicación que suma dos números. El programa `index.php` recoge los datos y el programa `suma.php` suma los dos números que recibe y muestra el resultado.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <h3>Calcula la suma de dos números</h3>
    <form action="suma.php" method="get">
      Primer número: <input type="number" name="a"><br>
      Segundo número: <input type="number" name="b"><br>
      <input type="submit" value="Sumar">
    </form>
  </body>
</html>
```

Fíjate que los campos `input` del formulario son de tipo `number`. Se podrían declarar de tipo `text` y el programa funcionaría sin problemas pero se declaran `number` para que el formulario no deje pasar los datos si se introduce algo que no sea un número. Imagínate que el usuario introduce la palabra `hola` en un campo en lugar de un número. Si el `input` es de tipo `text`, la palabra `hola` llegaría a `suma.php` y se intentaría sumar, lo que provocaría un error. Sin embargo, si el `input` es de tipo `number`, el formulario no envía `hola` sino que muestra un mensaje diciéndole al usuario que introduzca un número.

En la medida de lo posible, intenta utilizar siempre el tipo adecuado en los campos `input`; por ejemplo `number`, `color`, `date`, `email`, `url`, etc. de tal forma que los datos que envía el formulario tengan el formato correcto.

A continuación se muestra el fichero `suma.php`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $a = $_GET['a'];
      $b = $_GET['b'];
      echo "La suma de $a mas $b es ", $a + $b;
    ?>
  </body>
</html>
```


Como hemos visto anteriormente, en PHP no hace falta declarar las variables especificando el tipo. Una misma variable puede contener incluso valores de distintos tipos durante su vida útil. Esto hace que no necesitemos ninguna conversión como habría que hacer utilizando otros lenguajes.

2.2 Métodos get y post

Mediante el atributo `method` de la etiqueta `form` se debe especificar un método de envío; los dos métodos posibles son `get` y `post`. El resultado final es el mismo, la única diferencia radica en que con el método `get` se pueden ver los parámetros que envía el formulario en la barra de direcciones del navegador.

2.3 Ejercicios



Ejercicio 1

Realiza un programa que pida dos números y luego muestre el resultado de su multiplicación.



Ejercicio 2

Realiza un conversor de euros a pesetas. Ahora la cantidad en euros que se quiere convertir se deberá introducir por teclado.



Ejercicio 3

Realiza un conversor de pesetas a euros. La cantidad en pesetas que se quiere convertir se deberá introducir por teclado.



Ejercicio 4

Escribe un programa que sume, reste, multiplique y divida dos números introducidos por teclado.



Ejercicio 5

Escribe un programa que calcule el área de un rectángulo.



Ejercicio 6

Escribe un programa que calcule el área de un triángulo.



Ejercicio 7

Escribe un programa que calcule el total de una factura a partir de la base imponible.



Ejercicio 8

Escribe un programa que calcule el salario semanal de un trabajador en base a las horas trabajadas. Se pagarán 12 euros por hora.



Ejercicio 9

Escribe un programa que calcule el volumen de un cono mediante la fórmula $V = \frac{1}{3}\pi r^2 h$.



Ejercicio 10

Realiza un conversor de Mb a Kb.



Ejercicio 11

Realiza un conversor de Kb a Mb.

3. Sentencia condicional (if y switch)

Las sentencias condicionales permiten bifurcar el flujo del programa en función del cumplimiento o no de una o varias condiciones.

3.1 Sentencia if

La sentencia if permite la ejecución de una serie de instrucciones dependiendo del resultado de evaluar una expresión lógica¹. Se podría traducir al español como “si se cumple esta condición haz esto y si no, haz esto otro”.

El formato del if es el siguiente:

```
if (condición) {  
  
    sentencias a ejecutar cuando la condición es cierta  
  
} else {  
  
    Sentecias a ejecutar cuando la condición es falsa.  
  
}
```

A continuación se muestra un ejemplo completo en PHP:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
    <?php  
      $a = 8;  
      $b = 3;  
      if ($a < $b) {  
        echo "a es menor que b";  
      } else {  
        echo "a no es menor que b";  
      }  
    </?php>  
  </body>  
</html>
```

¹El resultado de evaluar una expresión lógica siempre es verdadero o falso.

```
    ?>
</body>
</html>
```

En este ejemplo, la condición no es verdadera por lo que se ejecuta la parte de código correspondiente al `else`.

En PHP también existe la posibilidad de utilizar una condición con la pareja de símbolo `?` y `:` al estilo de C. El formato es el siguiente:

```
(condición) ? expresión1 : expresión2
```

El funcionamiento es el siguiente: se evalúa la condición; si la condición es verdadera, el resultado que se devuelve es `expresión1` y si la condición es falsa, se devuelve `expresión2`.

A continuación se muestra un programa que ilustra este tipo de sentencia.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      $x = (20 > 6) ? 11 : 22;
      echo $x;
    ?>
  </body>
</html>
```

La expresión `(20 > 6)` es verdadera, por tanto en la variable `$x` se almacena el número 11 que es lo que se muestra por pantalla. Prueba a cambiar la condición para que sea falsa para comprobar que, esta vez, se muestra el 22.

3.2 Operadores de comparación

En el ejemplo de la sección anterior se utiliza el operador `<` en la comparación `if ($a < $b)` para saber si el valor de la variable `$a` es menor que el de la variable `$b`. Existen más operadores de comparación, a continuación se muestran todos ellos en una tabla:

Operador	Nombre	Ejemplo	Devuelve verdadero cuando...
==	Igual	\$a == \$b	\$a es igual \$b (aunque sean de diferente tipo)
===	Igual	\$a === \$b	\$a es igual \$b (y además son del mismo tipo)
!=	Distinto	\$a != \$b	\$a es distinto \$b
<	Menor que	\$a < \$b	\$a es menor que \$b
>	Mayor que	\$a > \$b	\$a es mayor que \$b
<=	Menor o igual	\$a <= \$b	\$a es menor o igual que \$b
>=	Mayor o igual	\$a >= \$b	\$a es mayor o igual que \$b

En el siguiente programa se muestra claramente la diferencia entre == y ===.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      $a = 5;
      $b = "5";
      echo ($a == 5)?"verdadero":"falso", "<br>";
      echo ($a === 5)?"verdadero":"falso", "<br>";
      echo ($b == 5)?"verdadero":"falso", "<br>";
      echo ($b === 5)?"verdadero":"falso", "<br>";
    ?>
  </body>
</html>
```

El resultado del programa sería el siguiente:

```
verdadero
verdadero
verdadero
falso
```

3.3 Operadores lógicos

Las comparaciones se pueden combinar mediante los operadores lógicos. Por ejemplo, si queremos saber si la variable \$a es mayor que \$b y además menor que \$c, escribiríamos `if (($a > $b) && ($a < $c))`. La siguiente tabla muestra los operadores lógicos de PHP:

Operador	Nombre	Ejemplo	Devuelve verdadero cuando...
&&	Y	(7>2) && (2<4)	Devuelve verdadero cuando ambas condiciones son verdaderas.
and	Y	(7>2) and (2<4)	Devuelve verdadero cuando ambas condiciones son verdaderas.
	O	(7>2) (2<4)	Devuelve verdadero cuando al menos una de las dos es verdadera.
or	O	(7>2) or (2<4)	Devuelve verdadero cuando al menos una de las dos es verdadera.
!	No	!(7>2)	Niega el valor de la expresión.

A continuación se muestra el uso de estos operadores lógicos en un programa PHP:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $a = 8;
      $b = 3;
      $c = 3;
      echo ($a == $b) && ($c > $b), "<br>";
      echo ($a == $b) || ($b == $c), "<br>";
      echo !($b <= $c), "<br>";
    ?>
  </body>
</html>
```

Observa que cuando el resultado de evaluar la expresión lógica es verdadero, se muestra un 1 por pantalla.

3.4 Sentencia switch

En ocasiones es necesario comparar el valor de una variable con una serie de valores concretos. Es algo parecido (aunque no exactamente igual) a utilizar varios if consecutivos.

El formato del switch es el siguiente:

```
switch(variable) {  
  
    case valor1:  
        sentencias  
        break;  
  
    case valor2:  
        sentencias  
        break;  
    .  
    .  
    .  
  
    default:  
        sentencias  
}
```

A continuación se muestra un ejemplo completo en PHP:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="UTF-8">  
    <title></title>  
  </head>  
  <body>  
    <?php  
      $posicion = "arriba";  
  
      switch($posicion) {  
        case "arriba": // Bloque 1  
          echo "La variable contiene";  
          echo " el valor arriba";  
          break;  
        case "abajo": // Bloque 2  
          echo "La variable contiene";  
          echo " el valor abajo";  
          break;  
        default: // Bloque 3  
          echo "La variable contiene otro valor";  
          echo " distinto de arriba y abajo";  
      }  
    ?>  
  </body>  
</html>
```


Prueba a cambiar el valor de la variable `$posicion` del ejemplo anterior y observa cómo se ejecuta el bloque de sentencias correspondiente.

3.5 Carga reiterada de una página. Adivina el número.

Hemos visto anteriormente que la recogida de datos se realiza mediante un formulario contenido en una página web. El sistema habitual consiste en tener una página con nombre `index.php` o `index.html` que recoge los datos necesarios mediante un formulario y que manda esos datos a otra página que los procesa. ¿Qué pasa entonces cuando tenemos que enviar datos una y otra vez y realizar cálculos según esos nuevos datos con un resultado diferente en cada ocasión? La respuesta es fácil: los datos se mandan a la misma página que los envía todas las veces que haga falta.

Lo entenderemos mejor con un ejemplo. Vamos a realizar un pequeño juego con nombre “Adivina el número”. El usuario debe adivinar un número entre 0 y 100; el programa irá guiando al usuario diciendo si el número introducido es mayor o menor que el que está pensando el ordenador. Tendremos en primer lugar una página llamada `index.php` que se utiliza únicamente para inicializar las variables:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    Adivina el número que estoy pensando.
    <form action="adivina.php" method="post">
      <input type="hidden" name="numeroIntroducido" value="555">
      <input type="hidden" name="oportunidades" value="5">
      <input type="submit" value="Continuar">
    </form>
  </body>
</html>
```

Fíjate que se inicializa `oportunidades` a 5; ese será el número de oportunidades que tendrá el usuario para adivinar el número. La variable `numeroIntroducido` se inicializa de forma arbitraria a 555 simplemente para indicarle a la página principal del juego cuándo se ejecuta por primera vez. Este número se puede cambiar pero debe quedar siempre fuera del intervalo [0 - 100] para que no se confunda con alguno de los números introducidos por el usuario.

A continuación se muestra la página `adivina.php` que constituye el cuerpo principal del programa:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $oportunidades = $_POST['oportunidades'];
      $numeroIntroducido = $_POST['numeroIntroducido'];

      $numeroSecreto = 24;

      if ($numeroIntroducido == $numeroSecreto) {
        echo "Enhorabuena, has acertado el número.";
      } else {
        if ($oportunidades == 0) {
          echo "Lo siento, has agotado todas tus oportunidades. Has perdido";
        } else {
          if ($numeroIntroducido!=555) {
            if ($numeroSecreto > $numeroIntroducido)
              echo "El número que estoy pensando es mayor que el número que has introducid\
o.<br>";
            else
              echo "El número que estoy pensando es menor que el número que has introducid\
o.<br>";
          }
        }
      }
    <?>
    Te quedan <?= $oportunidades-- ?> oportunidades para adivinar el número.<br>
    Introduce un número del 0 al 100
    <form action="adivina.php" method="post">
      <input type="text" name="numeroIntroducido">
      <input type="hidden" name="oportunidades" value="<?= $oportunidades ?>">
      <input type="submit" value="Continuar">
    </form>
    <?php
  }
}
?>
</body>
</html>

```

3.6 Ejercicios



Ejercicio 1

Escribe un programa que pida por teclado un día de la semana y que diga qué asignatura toca a primera hora ese día.



Ejercicio 2

Realiza un programa que pida una hora por teclado y que muestre luego buenos días, buenas tardes o buenas noches según la hora. Se utilizarán los tramos de 6 a 12, de 13 a 20 y de 21 a 5, respectivamente. Sólo se tienen en cuenta las horas, los minutos no se deben introducir por teclado.



Ejercicio 3

Escribe un programa en que dado un número del 1 a 7 escriba el correspondiente nombre del día de la semana.



Ejercicio 4

Vamos a ampliar uno de los ejercicios de la relación anterior para considerar las horas extras. Escribe un programa que calcule el salario semanal de un trabajador teniendo en cuenta que las horas ordinarias (40 primeras horas de trabajo) se pagan a 12 euros la hora. A partir de la hora 41, se pagan a 16 euros la hora.



Ejercicio 5

Realiza un programa que resuelva una ecuación de primer grado (del tipo $ax + b = 0$).



Ejercicio 6

Realiza un programa que calcule el tiempo que tardará en caer un objeto desde una altura h . Aplica la fórmula $t = \sqrt{\frac{2h}{g}}$ siendo $g = 9.81m/s^2$



Ejercicio 7

Realiza un programa que calcule la media de tres notas.



Ejercicio 8

Amplía el programa anterior para que diga la nota del boletín (insuficiente, suficiente, bien, notable o sobresaliente).



Ejercicio 9

Realiza un programa que resuelva una ecuación de segundo grado (del tipo $ax^2 + bx + c = 0$).



Ejercicio 10

Escribe un programa que nos diga el horóscopo a partir del día y el mes de nacimiento.



Ejercicio 11

Escribe un programa que dada una hora determinada (horas y minutos), calcule los segundos que faltan para llegar a la medianoche.



Ejercicio 12

Realiza un minicuestionario con 10 preguntas tipo test sobre las asignaturas que se imparten en el curso. Cada pregunta acertada sumará un punto. El programa mostrará al final la calificación obtenida. Pásale el minicuestionario a tus compañeros y pídeles que lo hagan para ver qué tal andan de conocimientos en las diferentes asignaturas del curso.



Ejercicio 13

Escribe un programa que ordene tres números enteros introducidos por teclado.



Ejercicio 14

Realiza un programa que diga si un número introducido por teclado es par y/o divisible entre 5.



Ejercicio 15

Realiza un programa que nos diga si hay probabilidad de que nuestra pareja nos está siendo infiel. El programa irá haciendo preguntas que el usuario contestará con verdadero o falso. Puedes utilizar radio buttons. Cada pregunta contestada como verdadero sumará 3 puntos. Las preguntas contestadas con falso no suman puntos. Utiliza el fichero [test_infidelidad.txt](https://github.com/LuisJoseSanchez/aprende-php-con-ejercicios/blob/master/soluciones/03_SentenciaCondicional/test_infidelidad.txt)² para obtener las preguntas y las conclusiones del programa.

²https://github.com/LuisJoseSanchez/aprende-php-con-ejercicios/blob/master/soluciones/03_SentenciaCondicional/test_infidelidad.txt



Ejercicio 16

Escribe un programa que diga cuál es la última cifra de un número entero introducido por teclado.



Ejercicio 17

Escribe un programa que diga cuál es la primera cifra de un número entero introducido por teclado. Se permiten números de hasta 5 cifras.



Ejercicio 18

Realiza un programa que nos diga cuántos dígitos tiene un número entero que puede ser positivo o negativo. Se permiten números de hasta 5 dígitos.



Ejercicio 19

Realiza un programa que diga si un número entero positivo introducido por teclado es capicúa. Se permiten números de hasta 5 cifras.

4. Bucles

Los bucles se utilizan para repetir un conjunto de sentencias.

4.1 Bucle for

Se suele utilizar cuando se conoce previamente el número exacto de iteraciones que se van a realizar. La sintaxis es la siguiente:

```
for (expresion1 ; expresion2 ; expresion3) {  
  
    sentencias  
  
}
```

Justo al principio se ejecuta `expresion1`, normalmente se usa para inicializar una variable. El bucle se repite mientras se cumpla `expresion2`. En cada iteración del bucle se ejecuta `expresion3`, que suele ser el incremento o decremento de una variable. A continuación se muestra un ejemplo:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
    <?php  
      for($i = 0 ; $i < 10 ; $i++) {  
        echo "El valor de i es ", $i, "<br>";  
      }  
    ?>  
  </body>  
</html>
```

4.2 Bucle while

El bucle `while` se utiliza para repetir un conjunto de sentencias siempre que se cumpla una determinada condición. Es importante reseñar que la condición se comprueba al comienzo del bucle, por lo que se podría dar el caso de que dicho bucle no se ejecutase nunca. La sintaxis es la siguiente:

```
while (expresion) {  
  
    sentencias  
  
}
```

Las sentencias se ejecutan una y otra vez mientras *expresion* sea verdadera. El siguiente ejemplo produce la misma salida que el ejemplo anterior, muestra cómo cambian los valores de *\$i* del 0 al 9.

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
    <?php  
      $i = 0;  
      while( $i < 10 ) {  
        echo "El valor de i es ", $i, "<br>"; $i++;  
      }  
    ?>  
  </body>  
</html>
```

4.3 Bucle do-while

El bucle *do-while* funciona de la misma manera que el bucle *while*, con la salvedad de que *expresion* se evalúa al final de la iteración. Las sentencias que encierran el bucle *do-while*, por tanto, se ejecutan como mínimo una vez. La sintaxis es la siguiente:

```
do {  
  
    sentencias  
  
} while (expresion)
```

El siguiente ejemplo es el equivalente *do-while* a los dos ejemplos anteriores.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $i = 0;
      do {
        echo "El valor de i es ", $i, "<br>"; $i++;
      } while( $i < 10 )
    ?>
  </body>
</html>
```


4.4 Ejercicios



Ejercicio 1

Muestra los números múltiplos de 5 de 0 a 100 utilizando un bucle `for`.



Ejercicio 2

Muestra los números múltiplos de 5 de 0 a 100 utilizando un bucle `while`.



Ejercicio 3

Muestra los números múltiplos de 5 de 0 a 100 utilizando un bucle `do-while`.



Ejercicio 4

Muestra los números del 320 al 160, contando de 20 en 20 utilizando un bucle `for`.



Ejercicio 5

Muestra los números del 320 al 160, contando de 20 en 20 utilizando un bucle `while`.



Ejercicio 6

Muestra los números del 320 al 160, contando de 20 en 20 utilizando un bucle `do-while`.



Ejercicio 7

Realiza el control de acceso a una caja fuerte. La combinación será un número de 4 cifras. El programa nos pedirá la combinación para abrirla. Si no acertamos, se nos mostrará el mensaje “Lo siento, esa no es la combinación” y si acertamos se nos dirá “La caja fuerte se ha abierto satisfactoriamente”. Tendremos cuatro oportunidades para abrir la caja fuerte.



Ejercicio 8

Muestra la tabla de multiplicar de un número introducido por teclado. El resultado se debe mostrar en una tabla (`<table>` en HTML).



Ejercicio 9

Realiza un programa que nos diga cuántos dígitos tiene un número introducido por teclado.



Ejercicio 10

Escribe un programa que calcule la media de un conjunto de números positivos introducidos por teclado. A priori, el programa no sabe cuántos números se introducirán. El usuario indicará que ha terminado de introducir los datos cuando meta un número negativo.



Ejercicio 11

Escribe un programa que muestre en tres columnas, el cuadrado y el cubo de los 5 primeros números enteros a partir de uno que se introduce por teclado.



Ejercicio 12

Escribe un programa que muestre los n primeros términos de la serie de Fibonacci. El primer término de la serie de Fibonacci es 0, el segundo es 1 y el resto se calcula sumando los dos anteriores, por lo que tendríamos que los términos son 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144... El número n se debe introducir por teclado.



Ejercicio 13

Escribe un programa que lea una lista de diez números y determine cuántos son positivos, y cuántos son negativos.



Ejercicio 14

Escribe un programa que pida una base y un exponente (entero positivo) y que calcule la potencia.



Ejercicio 15

Escribe un programa que dados dos números, uno real (base) y un entero positivo (exponente), saque por pantalla todas las potencias con base el número dado y exponentes entre uno y el exponente introducido. No se deben utilizar funciones de exponenciación. Por ejemplo, si introducimos el 2 y el 5, se deberán mostrar 2^1 , 2^2 , 2^3 , 2^4 , 2^5 .



Ejercicio 16

Escribe un programa que diga si un número introducido por teclado es o no primo. Un número primo es aquel que sólo es divisible entre él mismo y la unidad.



Ejercicio 17

Realiza un programa que sume los 100 números siguientes a un número entero y positivo introducido por teclado. Se debe comprobar que el dato introducido es correcto (que es un número positivo).



Ejercicio 18

Escribe un programa que obtenga los números enteros comprendidos entre dos números introducidos por teclado y validados como distintos, el programa debe empezar por el menor de los enteros introducidos e ir incrementando de 7 en 7.



Ejercicio 19

Realiza un programa que pinte una pirámide por pantalla. La altura se debe pedir por teclado mediante un formulario. La pirámide estará hecha de bolitas, ladrillos o cualquier otra imagen de las 5 que se deben dar a elegir mediante un formulario.



Ejercicio 20

Igual que el ejercicio anterior pero esta vez se debe pintar una pirámide hueca.



Ejercicio 21

Realiza un programa que vaya pidiendo números hasta que se introduzca un número negativo y nos diga cuantos números se han introducido, la media de los impares y el mayor de los pares. El número negativo sólo se utiliza para indicar el final de la introducción de datos pero no se incluye en el cómputo.



Ejercicio 22

Muestra por pantalla todos los números primos entre 2 y 100, ambos incluidos.



Ejercicio 23

Escribe un programa que permita ir introduciendo una serie indeterminada de números hasta que la suma de ellos supere el valor 10000. Cuando esto último ocurra, se debe mostrar el total acumulado, el contador de los números introducidos y la media.



Ejercicio 24

Escribe un programa que lea un número N e imprima una pirámide de números con N filas como en la siguiente figura. Recuerda utilizar un tipo de letra de ancho fijo como por ejemplo Courier para que los espacios tengan la misma anchura que los números. 1

**Ejercicio 25**

Realiza un programa que pida un número por teclado y que luego muestre ese número al revés.

**Ejercicio 26**

Realiza un programa que pida primero un número y a continuación un dígito. El programa nos debe dar la posición (o posiciones) contando de izquierda a derecha que ocupa ese dígito en el número introducido.

**Ejercicio 27**

Escribe un programa que muestre, cuente y sume los múltiplos de 3 que hay entre 1 y un número leído por teclado.

**Ejercicio 28**

Escribe un programa que calcule el factorial de un número entero leído por teclado.

**Ejercicio 29**

Escribe un programa que muestre por pantalla todos los números enteros positivos menores a uno leído por teclado que no sean divisibles entre otro también leído de igual forma.

5. Arrays

5.1 Arrays clásicas

Un array es un tipo de dato capaz de almacenar múltiples valores. Se utiliza cuando tenemos muchos datos parecidos, por ejemplo, para almacenar la temperatura media diaria en Málaga durante el último año podríamos utilizar las variables \$temp0, \$temp1, \$temp2, \$temp3, \$temp4, ... y así hasta 365 variables distintas pero sería poco práctico; es mejor utilizar un array de nombre \$temp y usar un índice para referirnos a una temperatura concreta.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $temp[0] = 16;
      $temp[1] = 15;
      $temp[2] = 17;
      $temp[3] = 15;
      $temp[4] = 16;
      echo "La temperatura en Málaga el cuarto día del año fue de ";
      echo $temp[3], "°C";
    ?>
  </body>
</html>
```

Los valores de un array se pueden asignar directamente en una línea. El índice comienza en 0.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $temp = array(16, 15, 17, 15, 16);
      echo "La temperatura en Málaga el cuarto día del año fue de ";
      echo $temp[3], "°C";
    ?>
  </body>
</html>
```

```

    </body>
</html>

```

En este otro ejemplo, asignamos valores aleatorios a los elementos de un array.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      echo "<b>Notas:</b><br>";
      for ($i = 0; $i < 10; $i++) {
        // números aleatorios entre 0 y 10 (ambos incluidos)
        $n[$i] = rand(0, 10);
      }

      foreach ($n as $elemento) {
        echo $elemento, "<br>";
      }
    ?>
  </body>
</html>

```

A partir de PHP 5.4 se puede utilizar la sintaxis abreviada (utilizando corchetes) para definir un array.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $color = ["verde", "amarillo", "rojo", "azul", "blanco", "gris"];
      echo "Mañana me pongo una camiseta de color ", $color[rand(0, 5)], ".";
    ?>
  </body>
</html>

```

Como puedes ver en los ejemplos anteriores, no es necesario definir previamente un array antes de utilizarlo. Tampoco hay límite en cuanto al número de elementos que se pueden añadir al mismo. No obstante, se pueden crear arrays de tamaño fijo con `SplFixedArray` que son más eficientes en cuanto a uso de la memoria y más rápidas en las operaciones de lectura y escritura.

La función `var_dump()` se utiliza para mostrar el tipo y el valor de un dato, en este caso muestra los tipos y valores de cada uno de los elementos del array.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $a = new SplFixedArray(10);

      $a[0] = 843;
      $a[2] = 11;
      $a[6] = 1372;

      // Los valores del array que no se han inicializado son NULL
      foreach ($a as $elemento) {
        var_dump($elemento);
        echo "<br>";
      }
    ?>
  </body>
</html>

```

5.2 Arrays asociativos

En un array asociativo se pueden utilizar índices que no son numéricos, a modo de claves. Veamos un ejemplo de un array asociativo que almacena alturas (en centímetros) y que como índice o clave utiliza nombres.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $estatura = array("Rosa" => 168, "Ignacio" => 175, "Daniel" => 172, "Rubén" => 182);
      echo "La estatura de Daniel es ", $estatura['Daniel'] , " cm";
    ?>
  </body>
</html>

```

También se pueden asignar los valores de forma individual:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $estatura['Rosa'] = 168;
      $estatura['Ignacio'] = 175;
      $estatura['Daniel'] = 172;
      $estatura['Rubén'] = 182;
      echo "La estatura de Daniel es ", $estatura['Daniel'] , " cm";
    ?>
  </body>
</html>
```

Si el lector es suficientemente perspicaz ya se habrá dado cuenta de que `$_GET` y `$_POST` son arrays asociativos. Con los arrays asociativos se puede usar también la sintaxis abreviada que emplea corchetes.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $estatura = [
        "Rosa" => 168,
        "Ignacio" => 175,
        "Daniel" => 172,
        "Rubén" => 182,
      ];
      echo "La estatura de Daniel es ", $estatura['Daniel'] , " cm";
    ?>
  </body>
</html>
```

5.3 Arrays bidimensionales

Un array bidimensional utiliza dos índices para localizar cada elemento. Podemos ver este tipo de datos como un array que, a su vez, contiene otros arrays. En el siguiente ejemplo se define un array con 4 elementos que, a su vez, son un array asociativo cada uno de ellos.


```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <?php
      $persona = array (
        array( "nombre" => "Rosa", "estatura" => 168, "sexo" => "F"),
        array( "nombre" => "Ignacio", "estatura" => 175, "sexo" => "M"),
        array( "nombre" => "Daniel", "estatura" => 172, "sexo" => "M"),
        array( "nombre" => "Rubén", "estatura" => 182, "sexo" => "M")
      );

      echo "<b>DATOS SOBRE EL PERSONAL<b><br><hr>";

      $numPersonas = count($persona);

      for ($i = 0; $i < $numPersonas; $i++) {
        echo "Nombre: <b>", $persona[$i]['nombre'], "</b><br>";
        echo "Estatura: <b>", $persona[$i]['estatura'], " cm</b><br>";
        echo "Sexo: <b>", $persona[$i]['sexo'], "</b><br><hr>";
      }
    ?>
  </body>
</html>
```

Observa que la función `count()` permite saber el número de elementos de un array.

5.4 Iterador foreach

El iterador `foreach` se utiliza para recorrer todos los elementos de un array sin que tengamos que preocuparnos por los índices ni por el tamaño del array.

Es común cometer errores al utilizar arrays por no establecer bien el valor inicial o el valor final en el bucle que la recorre, o por no determinar bien el tamaño. Con `foreach` nos podemos despreocupar de todo eso, simplemente recorreremos todo el array de principio a fin. Veamos un ejemplo.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $cajonDeSastre = [30, -7, "Me gusta el queso", 56, "ieh!", 237];

      foreach ($cajonDeSastre as $cosa) {
        echo "$cosa<br>";
      }
    ?>
  </body>
</html>
```

5.5 Cómo recoger datos para un array mediante un formulario

Imagina que quieres pedir diez números por teclado y quieres guardar esos números en un array. Se puede pedir un número mediante un formulario, a continuación el siguiente, luego el siguiente, etc. pero ¿cómo enviarlos? Hay un truco muy sencillo. Se pueden ir concatenando valores en una cadena de caracteres y el resultado de esa concatenación se puede reenviar una y otra vez en un formulario como campo oculto. Por último, se puede utilizar la función `explode()` para convertir la cadena de caracteres en un array.

Es importante señalar que los valores que se van concatenando deben tener algún tipo de separación dentro de la cadena, por ejemplo un espacio en blanco.

A continuación se muestra un ejemplo completo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $n = $_GET['n'];
      $contadorNumeros = $_GET['contadorNumeros'];
      $numeroTexto = $_GET['numeroTexto'];

      if (!isset($n)) {
        $contadorNumeros = 0;
        $numeroTexto = "";
      }

      // Muestra los números introducidos
```

```

if ($contadorNumeros == 6) {
    $numeroTexto = $numeroTexto . " " . $n; // añade el último número leído
    $numeroTexto = substr($numeroTexto, 2); // quita los dos primeros
                                           // espacios de la cadena

    $numero = explode(" ", $numeroTexto);
    echo "Los números introducidos son: ";
    foreach ($numero as $n) {
        echo $n, " ";
    }
}

// Pide número y añade el actual a la cadena
if (($contadorNumeros < 6) || (!isset($n))) {
    ?>
    <form action="#" method="get">
        Introduzca un número:
        <input type="number" name="n" autofocus>
        <input type="hidden" name="contadorNumeros" value="<?= ++$contadorNumeros ?>">
        <input type="hidden" name="numeroTexto" value="<?= $numeroTexto . " " . $n ?>">
        <input type="submit" value="OK">
    </form>
    <?php
}
?>
</body>
</html>

```

5.6 Ejercicios



Ejercicio 1

Define tres arrays de 20 números enteros cada una, con nombres “numero”, “cuadrado” y “cubo”. Carga el array “numero” con valores aleatorios entre 0 y 100. En el array “cuadrado” se deben almacenar los cuadrados de los valores que hay en el array “numero”. En el array “cubo” se deben almacenar los cubos de los valores que hay en “numero”. A continuación, muestra el contenido de los tres arrays dispuesto en tres columnas.



Ejercicio 2

Escribe un programa que pida 10 números por teclado y que luego muestre los números introducidos junto con las palabras “máximo” y “mínimo” al lado del máximo y del mínimo respectivamente.



Ejercicio 3

Escribe un programa que lea 15 números por teclado y que los almacene en un array. Rota los elementos de ese array, es decir, el elemento de la posición 0 debe pasar a la posición 1, el de la 1 a la 2, etc. El número que se encuentra en la última posición debe pasar a la posición 0. Finalmente, muestra el contenido del array.



Ejercicio 4

Escribe un programa que genere 100 números aleatorios del 0 al 20 y que los muestre por pantalla separados por espacios. El programa pedirá entonces por teclado dos valores y a continuación cambiará todas las ocurrencias del primer valor por el segundo en la lista generada anteriormente. Los números que se han cambiado deben aparecer resaltados de un color diferente.



Ejercicio 5

Realiza un programa que pida la temperatura media que ha hecho en cada mes de un determinado año y que muestre a continuación un diagrama de barras horizontales con esos datos. Las barras del diagrama se pueden dibujar a base de la concatenación de una imagen.



Ejercicio 6

Realiza un programa que pida 8 números enteros y que luego muestre esos números de colores, los pares de un color y los impares de otro.



Ejercicio 7

Escribe un programa que genere 20 números enteros aleatorios entre 0 y 100 y que los almacene en un array. El programa debe ser capaz de pasar todos los números pares a las primeras posiciones del array (del 0 en adelante) y todos los números impares a las celdas restantes. Utiliza arrays auxiliares si es necesario.



Ejercicio 8

Realiza un programa que pida 10 números por teclado y que los almacene en un array. A continuación se mostrará el contenido de ese array junto al índice (0 – 9) utilizando para ello una tabla. Seguidamente el programa pasará los primos a las primeras posiciones, desplazando el resto de números (los que no son primos) de tal forma que no se pierda ninguno. Al final se debe mostrar el array resultante.

Por ejemplo:

Array inicial									
0	1	2	3	4	5	6	7	8	9
20	5	7	4	32	9	2	14	11	6

Array final									
0	1	2	3	4	5	6	7	8	9
5	7	2	11	20	4	32	9	14	6



Ejercicio 9

Realiza un programa que pida 10 números por teclado y que los almacene en un array. A continuación se mostrará el contenido de ese array junto al índice (0 – 9). Seguidamente el programa pedirá dos posiciones a las que llamaremos “inicial” y “final”. Se debe comprobar que inicial es menor que final y que ambos números están entre 0 y 9. El programa deberá colocar el número de la posición inicial en la posición final, rotando el resto de números para que no se pierda ninguno. Al final se debe mostrar el array resultante.

Por ejemplo:

Array inicial									
0	1	2	3	4	5	6	7	8	9
20	5	7	4	32	9	2	14	11	6

Posición inicial: 3

Posición final: 7

Array final									
0	1	2	3	4	5	6	7	8	9
6	20	5	7	32	9	2	4	14	11



Ejercicio 10

Realiza un programa que escoja al azar 10 cartas de la baraja española y que diga cuántos puntos suman según el juego de la brisca. Emplea un array asociativo para obtener los puntos a partir del nombre de la figura de la carta. Asegúrate de que no se repite ninguna carta, igual que si las hubieras cogido de una baraja de verdad.



Ejercicio 11

Crea un mini-diccionario español-inglés que contenga, al menos, 20 palabras (con su traducción). Utiliza un array asociativo para almacenar las parejas de palabras. El programa pedirá una palabra en español y dará la correspondiente traducción en inglés.



Ejercicio 12

Realiza un programa que escoja al azar 5 palabras en español del mini-diccionario del ejercicio anterior. El programa pedirá que el usuario teclee la traducción al inglés de cada una de las palabras y comprobará si son correctas. Al final, el programa deberá mostrar cuántas respuestas son válidas y cuántas erróneas.



Ejercicio 13

Rellena un array bidimensional de 6 filas por 9 columnas con números enteros positivos comprendidos entre 100 y 999 (ambos incluidos). Todos los números deben ser distintos, es decir, no se puede repetir ninguno. Muestra a continuación por pantalla el contenido del array de tal forma que se cumplan los siguientes requisitos:

- Los números de las dos diagonales donde está el mínimo deben aparecer en color verde.
- El mínimo debe aparecer en color azul.
- El resto de números debe aparecer en color negro.



Ejercicio 14

Escribe un programa que, dada una posición en un tablero de ajedrez, nos diga a qué casillas podría saltar un alfil que se encuentra en esa posición. Indícalo de forma gráfica sobre el tablero con un color diferente para estas casillas donde puede saltar la figura. El alfil se mueve siempre en diagonal. El tablero cuenta con 64 casillas. Las columnas se indican con las letras de la “a” a la “h” y las filas se indican del 1 al 8.



Ejercicio 15

Realiza un programa que sea capaz de rotar todos los elementos de una matriz cuadrada una posición en el sentido de las agujas del reloj. La matriz debe tener 12 filas por 12 columnas y debe contener números generados al azar entre 0 y 100. Se debe mostrar tanto la matriz original como la matriz resultado, ambas con los números convenientemente alineados.

6. Funciones

6.1 Implementando funciones para reutilizar código

En programación es muy frecuente reutilizar código, es decir, usar código ya existente. Cuando una parte de un programa requiere una funcionalidad que ya está implementada en otro programa no tiene mucho sentido emplear tiempo y energía en implementarla otra vez.

Una función es un trozo de código que realiza una tarea muy concreta y que se puede incluir en cualquier programa cuando hace falta resolver esa tarea. Opcionalmente, las funciones aceptan una entrada (parámetros de entrada) y devuelven una salida.

Observa el siguiente ejemplo. Se trata de un programa que pide un número mediante un formulario y luego dice si el número introducido es o no es primo.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      $n = $_POST['n'];

      if (!isset($n)) {
        ?>
        Introduzca un número para saber si es primo o no.<br>
        <form action=numeroPrimo.php method="post">
          <input type="number" name="n" min="0" autofocus><br>
          <input type="submit" value="Aceptar">
        </form>
        <?php
      } else {
        $esPrimo = true;

        for ($i = 2; $i < $n; $i++) {
          if ($n % $i == 0) {
            $esPrimo = false;
          }
        }
      }
    }
  </body>
</html>
```



```

// El 0 y el 1 no se consideran primos
if (($n == 0) || ($n == 1)) {
    $esPrimo = false;
}

if ($esPrimo) {
    echo "El $n es primo.";
} else {
    echo "El $n no es primo.";
}
}
?>
</body>
</html>

```

Podemos intuir que la tarea de averiguar si un número es o no primo será algo que utilizaremos con frecuencia más adelante así que podemos aislar el trozo de código que realiza ese cometido para usarlo con comodidad en otros programas.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php

        // Programa principal //////////////////////////////////////

        $numero = $_POST['numero'];

        if (!isset($numero)) {
            ?>
            Introduzca un número para saber si es primo o no.<br>
            <form action=numeroPrimoConFuncion.php method="post">
                <input type="number" name="numero" min="0" autofocus><br>
                <input type="submit" value="Aceptar">
            </form>
            <?php
        } else {
            if (esPrimo($numero)) {
                echo "El $numero es primo.";
            } else {
                echo "El $numero no es primo.";
            }
        }
    }
}

```

```

    }
}

// Funciones //////////////////////////////////////

function esPrimo($n) {
    $esPrimo = true;

    for ($i = 2; $i < $n; $i++) {
        if ($n % $i == 0) {
            $esPrimo = false;
        }
    }

    // El 0 y el 1 no se consideran primos
    if (($n == 0) || ($n == 1)) {
        $esPrimo = false;
    }

    return $esPrimo;
}
?>
</body>
</html>

```

Cada función tiene una cabecera y un cuerpo. En el ejemplo anterior la cabecera es

```
function esPrimo($n)
```

La función `esPrimo()` toma como parámetro un número entero y devuelve un valor lógico (`true` o `false`). Observa que, a diferencia de otros lenguajes de programación fuertemente tipados como Java, en PHP no hace falta indicar el tipo de dato que devuelve la función ni el/los tipo/s de datos de los parámetros que se pasan.

6.2 Creación de bibliotecas de funciones

Por lo general y salvo casos puntuales, las funciones se suelen agrupar en ficheros. Estos ficheros de funciones se incluyen posteriormente en el programa principal mediante `include` o `include_once` seguido del nombre completo del fichero.

Veamos cómo utilizar la función `esPrimo()` desde un fichero independiente.

El siguiente código corresponde al fichero `matematicas.php`.

```

<?php

function esPrimo($n) {
    $esPrimo = true;

    for ($i = 2; $i < $n; $i++) {
        if ($n % $i == 0) {
            $esPrimo = false;
        }
    }

    // El 0 y el 1 no se consideran primos
    if (($n == 0) || ($n == 1)) {
        $esPrimo = false;
    }

    return $esPrimo;
}

```

El programa principal es el siguiente.

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
            // Carga las funciones matemáticas
            include 'matematicas.php';

            $numero = $_POST['numero'];

            if (!isset($numero)) {
                ?>
                Introduzca un número para saber si es primo o no.<br>
                <form action=numeroPrimo2.php method="post">
                    <input type="number" name="numero" min="0" autofocus><br>
                    <input type="submit" value="Aceptar">
                </form>
            } else {
                <?php
                if (esPrimo($numero)) {
                    echo "El $numero es primo.";
                }
            }

```

```

        } else {
            echo "El $numero no es primo.";
        }
    }
    ?>
</body>
</html>

```

6.3 ¿Se pueden sobrecargar las funciones en PHP?

En la mayoría de lenguajes de programación, las funciones se pueden sobrecargar. Esto significa que se pueden definir varias funciones con el mismo nombre pero con distinto número de parámetros, o bien con el mismo número de parámetros pero de distinto tipo.

En PHP no se pueden sobrecargar funciones, es decir, no podemos definir dos funciones con el mismo nombre aunque tengan distinto número de parámetros; pero sí se puede hacer “un apaño” para simular el comportamiento de la sobrecarga.

Fíjate en el siguiente código.

```

<?php
// Ejemplo de sobrecarga de un método según el
// número de parámetros que se pasan.
//
// Si se pasa un número, se devuelve el cuadrado;
// si se pasan dos, se devuelve la multiplicación
// y si se pasan tres, se devuelve la suma.
function opera($x, $y, $z) {
    if (!isset($y)) {
        return $x * $x;
    } else if (!isset($z)) {
        return $x * $y;
    } else {
        return $x + $y + $z;
    }
}

// Ejemplo de sobrecarga de un método según el
// tipo de los parámetros que se pasan.
//
// Si se pasan dos números enteros se devuelve
// la suma; en caso contrario se muestran los
// parametros separados por coma.
function opera2($a, $b) {
    if (is_int($a) && is_int($b)) {
        return $a + $b;
    }
}

```

```

    } else {
        return $a . ", " . $b;
    }
}

```

La función `opera()` se comporta como tres funciones en una sola. Si se le pasa un único número, la función devuelve el cuadrado de ese número; si se pasan dos parámetros, devuelve la multiplicación y si se pasan tres, devuelve la suma de todos ellos. Para comprobar si se pasa o no un determinado parámetro se utiliza la función `isset()`.

La función `opera2()` se comporta como dos funciones en una. Esta vez la simulación de la sobrecarga no se realiza en función del número de parámetros sino del tipo. Mediante `is_int()` podemos comprobar si un determinado parámetro es un número entero. En el caso que nos ocupa, si la función `opera2()` recibe como parámetros dos números enteros, se devuelve la suma; en caso contrario, lo que se devuelve es una cadena de caracteres con los parámetros que se han pasado separados por coma.

El programa principal que llama a las funciones es el siguiente.

Observa que en ambos ficheros se especifica que las clases declaradas (y por tanto las funciones que se definen dentro) pertenecen al paquete `matematicas` mediante la línea `package matematicas`. Ahora ya podemos probar las funciones desde un programa externo. El programa `PruebaFunciones.java` está fuera de la carpeta `matematicas`, justo en un nivel superior en la estructura de directorios.

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
            include_once 'funciones.php';

            // Mismo nombre de función con diferente
            // número de parámetros.
            echo opera(2). "<br>";
            echo opera(2, 3). "<br>";
            echo opera(2, 3, 10). "<br>";

            // Mismo nombre de función con distintos
            // tipos de parámetros.
            echo opera2(10, 20). "<br>";
            echo opera2("melón", "sandía"). "<br>";
        ?>

```

```
</body>  
</html>
```

La salida del programa anterior es la siguiente:

```
4  
6  
15  
30  
melón, sandía
```

6.4 Ejercicios



Ejercicios 1-14

Crea una biblioteca de funciones matemáticas que contenga las siguientes funciones. Recuerda que puedes usar unas dentro de otras si es necesario.

1. **esCapicua**: Devuelve verdadero si el número que se pasa como parámetro es capicúa y falso en caso contrario.
2. **esPrimo**: Devuelve verdadero si el número que se pasa como parámetro es primo y falso en caso contrario.
3. **siguientePrimo**: Devuelve el menor primo que es mayor al número que se pasa como parámetro.
4. **potencia**: Dada una base y un exponente devuelve la potencia.
5. **digitos**: Cuenta el número de dígitos de un número entero.
6. **voltea**: Le da la vuelta a un número.
7. **digitoN**: Devuelve el dígito que está en la posición n de un número entero. Se empieza contando por el 0 y de izquierda a derecha.
8. **posicionDeDigito**: Da la posición de la primera ocurrencia de un dígito dentro de un número entero. Si no se encuentra, devuelve -1.
9. **quitaPorDetras**: Le quita a un número n dígitos por detrás (por la derecha).
10. **quitaPorDelante**: Le quita a un número n dígitos por delante (por la izquierda).
11. **pegaPorDetras**: Añade un dígito a un número por detrás.
12. **pegaPorDelante**: Añade un dígito a un número por delante.
13. **trozoDeNumero**: Toma como parámetros las posiciones inicial y final dentro de un número y devuelve el trozo correspondiente.
14. **juntaNumeros**: Pega dos números para formar uno.



Ejercicio 15

Muestra los números primos que hay entre 1 y 1000.



Ejercicio 16

Muestra los números capicúa que hay entre 1 y 99999.



Ejercicio 17

Escribe un programa que pase de binario a decimal.



Ejercicio 18

Escribe un programa que pase de decimal a binario.



Ejercicio 19

Une y amplía los dos programas anteriores de tal forma que se permita convertir un número entre cualquiera de las siguientes bases: decimal, binario, hexadecimal y octal.



Ejercicios 20-28

Crea una biblioteca de funciones para arrays (de una dimensión) de números enteros que contenga las siguientes funciones:

1. **generaArrayInt**: Genera un array de tamaño *n* con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.
2. **minimoArrayInt**: Devuelve el mínimo del array que se pasa como parámetro.
3. **maximoArrayInt**: Devuelve el máximo del array que se pasa como parámetro.
4. **mediaArrayInt**: Devuelve la media del array que se pasa como parámetro.
5. **estaEnArrayInt**: Dice si un número está o no dentro de un array.
6. **posicionEnArray**: Busca un número en un array y devuelve la posición (el índice) en la que se encuentra.
7. **volteaArrayInt**: Le da la vuelta a un array.
8. **rotaDerechaArrayInt**: Rota *n* posiciones a la derecha los números de un array.
9. **rotaIzquierdaArrayInt**: Rota *n* posiciones a la izquierda los números de un array.



Ejercicio 29-34

Crea una biblioteca de funciones para arrays bidimensionales (de dos dimensiones) de números enteros que contenga las siguientes funciones:

1. **generaArrayBiInt**: Genera un array de tamaño *n* x *m* con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.
2. **filaDeArrayBiInt**: Devuelve la fila *i*-ésima del array que se pasa como parámetro.
3. **columnaDeArrayBiInt**: Devuelve la columna *j*-ésima del array que se pasa como parámetro.
4. **coordenadasEnArrayBiInt**: Devuelve la fila y la columna (en un array con dos elementos) de la primera ocurrencia de un número dentro de un array bidimensional. Si el número no se encuentra en el array, la función devuelve el array `{-1, -1}`.
5. **esPuntoDeSilla**: Dice si un número es o no punto de silla, es decir, mínimo en su fila y máximo en su columna.
6. **diagonal**: Devuelve un array que contiene una de las diagonales del array bidimensional que se pasa como parámetro. Se pasan como parámetros fila, columna y dirección. La fila y la columna determinan el número que marcará las dos posibles diagonales dentro del array. La dirección es una cadena de caracteres que puede ser "nose" o "neso". La cadena "nose" indica que se elige la diagonal que va del noroeste hacia el sureste, mientras que la cadena "neso" indica que se elige la diagonal que va del noreste hacia el suroeste.

7. Sesiones y cookies

7.1 Sesiones

Las sesiones se utilizan en PHP para guardar información en la memoria RAM. Esta información no se pierde si el usuario salta de una página a otra.

Hemos visto anteriormente cómo enviar datos entre dos páginas mediante un formulario, ahora imagina que visitas varias veces la misma página, que saltas a otra, que vuelves de nuevo a la primera; sería muy engorroso estar mandando datos constantemente mediante formularios entre unas páginas y otras. Mediante las sesiones se puede conservar o modificar la información que nos interese independientemente de la/s página/s que se vayan visitando. El valor que se almacena en la memoria está disponible mientras no se cierre la sesión.

Un uso típico de una sesión es el carrito de la compra de una tienda on-line. Podemos visitar todas las páginas que queramos de la tienda e ir añadiendo o quitando productos del carrito gracias a que esta información se graba en una sesión.

La sesión comienza con la función `session_start()` y debe colocarse siempre al principio, antes de mostrar cualquier cosa en el documento HTML.

El ejemplo más sencillo del uso de sesiones es un contador de visitas a una página.

```
<?php
    session_start(); // Inicio de sesión

    if(isset($_SESSION['visitas'])) {
        $_SESSION['visitas']++;
    } else {
        $_SESSION['visitas'] = 1;
    }
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
        <?php
            echo "Visitas: " . $_SESSION['visitas'];
        ?>
    </body>
</html>
```

Observa que si `$_SESSION['visitas']` no tiene ningún valor asignado, se crea a la vez que se le asigna el 1, ya que es la primera vez que se visita la página; en caso contrario, se incrementa en 1 su valor de modo que va contando las veces que se carga la página. Incluso si se cierra la pestaña del navegador y se vuelve a abrir otra, el número de visitas continúa donde se quedó.

Para volver a reiniciar el contador de visitas y, por tanto cerrar la sesión, es necesario cerrar el navegador y abrirlo de nuevo.

Dentro del código PHP podemos indicar que queremos cerrar una sesión mediante `session_destroy()`.

En el siguiente ejemplo tenemos dos variables que podemos incrementar, decrementar o poner a cero. El valor de esas variable se guarda en una sesión.

```
<?php
    session_start(); // Inicia la sesión

    // La primera vez que se carga la página, se inicializan
    // las variables de sesión a y b a cero.
    if(!isset($_SESSION['a'])) {
        $_SESSION['a'] = 0;
    }

    if(!isset($_SESSION['b'])) {
        $_SESSION['b'] = 0;
    }
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
            switch($_POST['accion']) {
                case "incA" :
                    $_SESSION['a']++;
                    break;
                case "decA" :
                    $_SESSION['a']--;
                    break;
                case "incB" :
                    $_SESSION['b']++;
                    break;
                case "decB" :
                    $_SESSION['b']--;
                    break;
            }
        ?>
    </body>
</html>
```

```

        case "cierra":
            session_destroy();
            header("refresh: 0;"); // refresca la página
    }
?>

<h1>
a = <?php echo $_SESSION['a'] ?><br>
b = <?php echo $_SESSION['b'];?>
</h1>

<form action="#" method="POST">
    <select name="accion">
        <option value="incA">Incrementa a</option>
        <option value="decA">Decrementa a</option>
        <option value="incB">Incrementa b</option>
        <option value="decB">Decrementa b</option>
        <option value="cierra">Cierra sesión</option>
    </select>
    <input type="submit" value="OK">
</form>
</body>
</html>

```

7.2 Cookies

Una cookie (galleta en inglés) es un fichero que se graba en el ordenador del propio usuario, no en el servidor. Permite guardar información de tal forma que no es necesario enviarla mediante formularios al pasar de una página a otra. Una cookie es algo parecido a una sesión aunque, a diferencia de esta última, la cookie se graba en el disco duro del ordenador.

Las cookies se crean con la función `setcookie()` que debe estar situada al comienzo de la página, antes que cualquier etiqueta HTML. El formato de esta función es el siguiente:

```
setcookie(nombre, valor, momento de expiración)
```

El momento de expiración es aquel en el que la cookie se elimina y se expresa en segundos. Normalmente se utiliza la función `time()` junto con el número de segundos que queremos que dure la cookie. Por ejemplo, si queremos crear una cookie de nombre `usuario`, inicializada a `Luis` y que dure una semana, escribiríamos lo siguiente:

```
setcookie("usuario", "Luis", time() + 7*24*60*60)
```

Si no se especifica el momento de expiración, la cookie durará hasta que se cierre el navegador.

Para recuperar el valor de cualquier cookie se utiliza el array `$_COOKIE`. Por ejemplo, para mostrar el valor de la cookie de nombre `usuario`, escribiríamos lo siguiente:

```
echo $_COOKIE["usuario"];
```

A la hora de depurar un programa, es muy útil mostrar todas las cookies. Podemos hacer esto con `print_r($_COOKIE)`.

A continuación se muestra un ejemplo completo. Se trata de una aplicación que muestra nuestra actriz y nuestro actor favorito. Podemos cambiar nuestras preferencias de tal forma que la información permanece en el disco, así se conservan los datos aunque se reinicie el navegador, e incluso aunque se apague y se vuelva a encender el ordenador.

```
<?php
// Si se envían datos desde el formulario de actores,
// se actualizan las cookies

if (isset($_POST["actriz"])) {
    $actriz = $_POST["actriz"];
    $actor = $_POST["actor"];
    setcookie("actriz", $actriz, time() + 3*24*3600);
    setcookie("actor", $actor, time() + 3*24*3600);
} else if (isset($_COOKIE["actriz"])) {
    $actriz = $_COOKIE["actriz"];
    $actor = $_COOKIE["actor"];
}

// Borrado de cookies y variables

if (isset($_POST["borraCookies"])) {
    setcookie("actriz", NULL, -1);
    setcookie("actor", NULL, -1);
    unset($actriz);
    unset($actor);
}

?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
            if (!isset($actriz)) {
                echo "No has elegido todavía a tus actores favoritos.<br>";
                echo "Utiliza el siguiente formulario para hacerlo.<br>";
            } else {
                echo "<h2>Actriz favorita: ".$actriz."</h2>";
            }
        </?php>
    </body>
</html>
```

```
        echo "<h2>Actor favorito: ".$actor."</h2>";
        echo "Introduce nuevos nombres si quieres cambiar tus preferencias.<br>";
    }
?>
<form action="#" method="post">
    Actriz: <input type="text" name="actriz"><br>
    Actor: <input type="text" name="actor"><br>
    <input type="submit" value="Aceptar">
</form>
<hr>

<form action="#" method="post">
    <input type="hidden" name="borraCookies" value="si">
    <input type="submit" value="Borrar cookies">
</form>
</body>
</html>
```

Hay que tener en cuenta algo muy importante en cuanto al comportamiento de las cookies que puede dar muchos quebraderos de cabeza: el valor de una cookie se actualiza en la siguiente carga de la página, no tiene el último valor que se le da sino el anterior.

Fíjate cómo se borra una cookie, se utiliza el mismo `setcookie` que se usa para darle un valor, pero esta vez dándole el valor `NULL`.

```
setcookie("actriz", NULL, -1);
```

Puedes ver todas las cookies almacenadas en el equipo. Para el navegador **Firefox**, selecciona **Editar** → **Preferencias** → **Privacidad** → **eliminar cookies de forma individual**. Puedes buscar las cookies por dominio, así que filtrando por **localhost**, podrás ver las que hemos creado con el programa de ejemplo.

7.3 Ejercicios



Ejercicio 1

Escribe un programa que calcule la media de un conjunto de números positivos introducidos por teclado. A priori, el programa no sabe cuántos números se introducirán. El usuario indicará que ha terminado de introducir los datos cuando meta un número negativo. Utiliza sesiones.



Ejercicio 2

Realiza un programa que vaya pidiendo números hasta que se introduzca un numero negativo y nos diga cuantos números se han introducido, la media de los impares y el mayor de los pares. El número negativo sólo se utiliza para indicar el final de la introducción de datos pero no se incluye en el cómputo. Utiliza sesiones.



Ejercicio 3

Escribe un programa que permita ir introduciendo una serie indeterminada de números mientras su suma no supere el valor 10000. Cuando esto último ocurra, se debe mostrar el total acumulado, el contador de los números introducidos y la media. Utiliza sesiones.



Ejercicio 4

Establece un control de acceso mediante nombre de usuario y contraseña para cualquiera de los programas de la relación anterior. La aplicación no nos dejará continuar hasta que iniciemos sesión con un nombre de usuario y contraseña correctos.



Ejercicio 5

Crea una tienda *on-line* sencilla con un catálogo de productos y un carrito de la compra. Un catálogo de cuatro o cinco productos será suficiente. De cada producto se debe conocer al menos la descripción y el precio. Todos los productos deben tener una imagen que los identifique. Al lado de cada producto del catálogo deberá aparecer un botón **Comprar** que permita añadirlo al carrito. Si el usuario hace clic en el botón **Comprar** de un producto que ya estaba en el carrito, se deberá incrementar el número de unidades de dicho producto. Para cada producto que aparece en el carrito, habrá un botón **Eliminar** por si el usuario se arrepiente y quiere quitar un producto concreto del carrito de la compra. A continuación se muestra una captura de pantalla de una posible solución.

APRENDE PHP CON EJERCICIOS

SOLUCIONES A LOS EJERCICIOS

6. Sesiones y Cookies

Tienda on-line *La Estilográfica*

Productos



Pelikan Souvèran M-1000
Precio: 545 €
[Comprar](#)



Parker Duofold International
Precio: 406 €
[Comprar](#)



Visconti Van Gogh
Precio: 180 €
[Comprar](#)

Carrito



Pelikan Souvèran M-1000
Precio: 545 €
Unidades: 2
[Eliminar](#)



Parker Duofold International
Precio: 406 €
Unidades: 1
[Eliminar](#)

Total: 1496 €



Ejercicio 6

Amplía el programa anterior de tal forma que se pueda ver el detalle de un producto. Para ello, cada uno de los productos del catálogo deberá tener un botón **Detalle** que, al ser accionado, debe llevar al usuario a la vista de detalle que contendrá una descripción exhaustiva del producto en cuestión. Se podrán añadir productos al carrito tanto desde la vista de listado como desde la vista de detalle.



Ejercicio 7

Escribe un programa que guarde en una cookie el color de fondo (propiedad `background-color`) de una página. Esta página debe tener únicamente algo de texto y un formulario para cambiar el color.



Ejercicio 8

Realiza un programa que escoja al azar 5 palabras en inglés de un mini-diccionario. El programa pedirá que el usuario teclee la traducción al español de cada una de las palabras y comprobará si son correctas. Al final, el programa deberá mostrar cuántas respuestas son válidas y cuántas erróneas. La aplicación debe tener una opción para introducir los pares de palabras (inglés - español) que se deben guardar en cookies; de esta forma, si de vez en cuando se dan de alta nuevas palabras, la aplicación puede llegar a contar con un número considerable de entradas en el mini-diccionario.



Ejercicio 9

Amplía el ejercicio 6 de tal forma que los productos que se pueden elegir para comprar se almacenen en cookies. La aplicación debe ofrecer, por tanto, las opciones de alta, baja y modificación de productos.

8. Acceso a bases de datos

8.1 Acceso a BBDD desde PHP

Desde una página con código escrito en PHP nos podemos conectar a una base de datos y ejecutar comandos en lenguaje SQL - para hacer consultas, insertar o modificar registros, crear tablas, dar permisos, etc.

PHP puede trabajar con la práctica totalidad de gestores de bases de datos que hay disponibles, tanto comerciales como de código abierto.

En este libro veremos tres maneras diferentes de acceder a una base de datos desde PHP:

mysql

La interfaz `mysql` permite acceder a bases de datos MySQL. Esta interfaz se considera obsoleta y se desaconseja su uso en aplicaciones nuevas. No obstante, se ha venido utilizando en multitud de aplicaciones y es muy recomendable conocerla.

mysqli

Se trata de una mejora de la interfaz `mysql` (la “i” viene de “*improved*”). Por ejemplo, la extensión `mysqli` añade ciertas funcionalidades que no tiene `mysql` como las transacciones, los procedimientos almacenados y las sentencias múltiples. En aplicaciones nuevas se recomienda usar esta API, o bien PDO.

PDO

PDO son las siglas de *PHP Data Objects* (Objetos de Datos de PHP). Proporciona una capa de abstracción de tal forma que los métodos utilizados para acceder a los datos son independientes del sistema gestor de bases de datos utilizado. En la práctica, permite cambiar de SGBD sin cambiar el código PHP.

8.2 Extensión mysql

Establecimiento de una conexión

La función `mysql_connect(máquina, usuario, contraseña)` intenta abrir una conexión con el servidor MySQL en el host especificado - en nuestro caso haremos pruebas en `localhost`, o sea, en nuestra propia máquina - y devuelve un identificador de dicha conexión (un número) si ha tenido éxito o 0 (false) en caso de que no se haya podido establecer la conexión.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $conexion = mysql_connect("localhost", "root", "root");
      if ($conexion) {
        echo "Se ha establecido una conexión con el servidor de bases de datos.";
      } else {
        echo "No se ha podido establecer conexión con el servidor de bases de datos.";
      }
    ?>
  </body>
</html>
```

Con `mysql_select_db()` podemos seleccionar una base de datos sobre la que trabajar.

Mediante el siguiente código nos conectamos a la base de datos que tiene por nombre banco y hacemos una consulta a la tabla empleado para mostrar los datos de uno de los empleados, concretamente del que tiene el DNI 13579.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $conexion = mysql_connect("localhost", "root", root);

      mysql_select_db("banco", $conexion);
      mysql_set_charset('utf8');

      $consulta = mysql_query('SELECT * FROM empleado WHERE dni="13579"', $conexion);

      echo "Nombre: " . mysql_result($consulta, 0, "nombre") . "<br>";
      echo "Cargo: " . mysql_result($consulta, 0, "cargo") . "<br>";
      echo "Sueldo: " . mysql_result($consulta, 0, "sueldo") . "€<br>";
    ?>
  </body>
</html>
```

Observa que `mysql_result()` accede en cada una de las líneas del programa a un campo concreto - nombre, cargo o sueldo - de la primera y única tupla que se obtiene (la tupla 0) al hacer la consulta a la tabla empleado.

Listado completo de una tabla

Ahora mostraremos un listado completo, es decir, una consulta que muestra todas las filas que componen una tabla. Para ello, debemos recorrer con un bucle todos los valores de la variable `$consulta` - donde están almacenados todos los valores en bruto - con `mysql_fetch_array()`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <h2>
      Base de datos <u>banco</u><br>
      Tabla <u>cliente</u><br>
    </h2>
    <?php
      $conexion = mysql_connect("localhost", "root", "root");
      mysql_select_db("banco", $conexion);
      mysql_set_charset('utf8');
      $consulta = mysql_query("SELECT dni, nombre, direccion, telefono FROM cliente", $con\
exion);
      ?>

      <table border="1">
        <tr>
          <td><b>DNI</b></td>
          <td><b>Nombre</b></td>
          <td><b>Dirección</b></td>
          <td><b>Teléfono</b></td>
        </tr>

        <?php
          while ($registro = mysql_fetch_array($consulta)){
            echo "<tr>";
            echo "<td>".$registro[dni]."</td>";
            echo "<td>".$registro[nombre]."</td>";
            echo "<td>".$registro[direccion]."</td>";
            echo "<td>".$registro[telefono]."</td>";
            echo "</tr>";
          }
        ?>
      </table>
      <br>
      Número de clientes: <?= mysql_num_rows($consulta) ?>
```

```

</body>
</html>

```

En esta ocasión el bucle `while` va evaluando la función `mysql_fetch_array()`, que devuelve un array con el contenido del registro actual (que se almacena en `$registro`) y avanza una posición en la lista de registros devueltos en la consulta SQL.

Búsqueda de un valor en una tabla

A continuación tenemos un programa que busca la coincidencia de un valor determinado con un campo de una tabla de la base de datos. En concreto, se buscan todos los clientes cuyo nombre contenga una cadena de caracteres previamente introducida en un formulario.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <form method="post" action="#">
      Buscar en el campo nombre de la tabla de clientes:
      <input type="text" name="buscar">
      <input type="submit" value="Buscar">
    </form>

    <?php
      if (isset($_POST['buscar'])) {
        $buscar = $_POST['buscar'];
        $conexion = mysql_connect("localhost", "root", "root");
        mysql_select_db("banco", $conexion);
        mysql_set_charset('utf8');
        $sql = "SELECT * FROM cliente WHERE nombre LIKE '%$buscar%' ORDER BY nombre";
        $resultado = mysql_query($sql, $conexion);
        ?>

        <table border = '1'>
          <tr>
            <td><b>DNI</b></td>
            <td><b>Nombre</b></td>
            <td><b>Dirección</b></td>
            <td><b>Teléfono</b></td>
          </tr>

          <?php
            do {

```

```

    ?>
    <tr>
        <td><?= $registro[dni] ?></td>
        <td><?= $registro[nombre] ?></td>
        <td><?= $registro[direccion] ?></td>
        <td><?= $registro[telefono] ?></td>
    </tr>
    <?php
    } while ($registro = mysql_fetch_array($resultado));
    }
    ?>
</table>
</body>
</html>

```

El corazón del programa es la sentencia SQL que se envía al servidor MySQL, y más concretamente la cláusula WHERE:

```
WHERE nombre LIKE '%$buscar%'
```

Con la sentencia LIKE buscamos cualquier ocurrencia de la cadena contenida en \$buscar, mientras que con los signos de porcentaje (%) indicamos el lugar de la coincidencia, por ejemplo, si hubiesemos escrito nombre LIKE '%\$buscar', buscaríamos cualquier ocurrencia al final del campo nombre, mientras que si hubiesemos puesto nombre LIKE '\$buscar%', buscaríamos cualquier ocurrencia al principio del campo nombre.

Tratamiento de errores

Imagina que un programa PHP que hace uso de bases de datos no funciona. ¿Dónde puede estar el error? ¿en la conexión con el servidor? ¿en la selección de la base de datos? ¿en la consulta SQL? Para depurar correctamente este tipo de programas debemos tener en cuenta que cuando una operación MySQL se realiza de forma satisfactoria se devuelve true y en caso contrario se devuelve false.

Es recomendable por tanto ir comprobando que todas las operaciones se van realizando correctamente y, en caso contrario, mostrar un mensaje de error. El formato sería el siguiente:

```

if (!operacionMySQL) {
    echo "Se ha producido un error de MySQL";
}

```

Por ejemplo, para seleccionar una base de datos escribiríamos lo siguiente:

```
if (!mysql_select_db("banco", $conexion)) {
    echo "Se ha producido un error al seleccionar la base de datos.";
}
```

Cuando se produce un error que impide acceder correctamente a una base de datos, no tiene mucho sentido continuar con la ejecución del programa. Para detener el programa se puede utilizar `die("mensaje")`. Además, podemos acceder al último error que ha generado PHP mediante `mysql_error()`. A continuación se muestra un ejemplo:

```
if (!mysql_select_db("banco", $conexion)) {
    die("Error al seleccionar la base de datos: ".mysql_error());
}
```

8.3 Extensión mysqli

La extensión `mysqli` se puede utilizar con el formato procedural igual que `mysql` o bien con el formato de Programación Orientada a Objetos. En este libro utilizaremos la segunda opción.

Establecimiento de una conexión

Mediante `new mysqli(máquina, usuario, contraseña)` se crea una nueva conexión con el servidor MySQL en el host que se indica y con el nombre de usuario y contraseña especificados. Si la conexión es satisfactoria, `$conexion->connect_errno` tiene el valor 0; en caso contrario, `$conexion->connect_errno` tendrá un valor mayor que 0 (un código de error).

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
        <?php
            $conexion = new mysqli("localhost", "root", "root");
            if ($conexion->connect_errno > 0) {
                echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
                die("Error: " . $conexion->connect_error);
            } else {
                echo "Se ha establecido una conexión con el servidor de bases de datos.";
            }
        ?>
    </body>
</html>
```

Una vez establecida la conexión con el servidor, veamos cómo se realiza una consulta a la base de datos.

El programa que se muestra a continuación se conecta a la base de datos banco y hace una consulta a la tabla empleado para mostrar los datos del empleado que tiene el DNI 13579. Por tanto hace lo mismo que el ejemplo visto en la sección anterior, pero en esta ocasión utilizando la extensión `mysqli`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $conexion = new mysqli("localhost", "root", "root");
      if ($conexion->connect_errno > 0) {
        echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
        die ("Error: " . $conexion->connect_error);
      } else {
        $conexion->select_db("banco");
        $conexion->set_charset("utf8");

        $consulta = $conexion->query('SELECT * FROM empleado WHERE dni="13579"');

        $empleado = $consulta->fetch_object();

        echo "Nombre: " . $empleado->nombre . "<br>";
        echo "Cargo: " . $empleado->cargo . "<br>";
        echo "Sueldo: " . $empleado->sueldo . "€<br>";
      }
    ?>
  </body>
</html>
```

Mediante `$conexion->select_db("banco")` se selecciona la base de datos banco.

La siguiente línea se encarga de realizar una consulta a la base de datos, dejando el resultado en la variable `$consulta`.

```
$consulta = $conexion->query('SELECT * FROM empleado WHERE dni="13579"');
```

Para extraer los datos de `$consulta` en forma de objeto se utiliza `fetch_object()`.

Listado completo de una tabla

Igual que en la sección anterior, ahora mostraremos un listado completo con todas las filas que componen una tabla. Para ello iremos sacando todos los datos con `fetch_object()`.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <h2>
      Base de datos <u>banco</u><br>
      Tabla <u>cliente</u><br>
    </h2>
    <?php
      $conexion = new mysqli("localhost", "root", "root");
      if ($conexion->connect_errno > 0) {
        echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
        die ("Error: " . $conexion->connect_error);
      } else {
        $conexion->select_db("banco");
        $conexion->set_charset("utf8");

        $consulta = $conexion->query("SELECT dni, nombre, direccion, telefono FROM cliente\
");

        ?>
        <table border="1">
          <tr>
            <td><b>DNI</b></td>
            <td><b>Nombre</b></td>
            <td><b>Dirección</b></td>
            <td><b>Teléfono</b></td>
          </tr>

          <?php
            while ($cliente = $consulta->fetch_object()){
              ?>
              <tr>
                <td><?= $cliente->dni ?></td>
                <td><?= $cliente->nombre ?></td>
                <td><?= $cliente->direccion ?></td>
                <td><?= $cliente->telefono ?></td>
              </tr>
            <?php
          }
        }
        $conexion->close();
      ?>
    </?php
  </body>
</html>

```



```

    </body>
</html>

```

En la condición del bucle `while` se evalúa la expresión `$cliente = $consulta->fetch_object()`. Cuando no quedan más datos por extraer, esta expresión devuelve `false` y se termina la ejecución del bucle.

8.4 PDO (*PHP Data Objects*)

Establecimiento de una conexión

Para crear una conexión nueva a un servidor, se utiliza el constructor PDO de la siguiente manera:

```
$conexion = new PDO("mysql:host=localhost", "root", "root");
```

Observa que, en primer lugar, hay que indicar el gestor de bases de datos que se utilizará. En este caso es `mysql`. Una vez establecida la conexión, cualquier operación que se realice sobre una base de datos se efectuará con métodos que son independientes del SGBD; es decir, no cambiará el código aunque se cambie el SGBD.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      try {
        $conexion = new PDO("mysql:host=localhost", "root", "root");
        echo "Se ha establecido una conexión con el servidor de bases de datos.";
      } catch (PDOException $e) {
        echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
        die ("Error: " . $e->getMessage());
      }
    ?>
  </body>
</html>

```

Para efectuar una conexión a otro gestor de bases de datos, hay que cambiar `mysql` dentro de la sentencia

```
$conexion = new PDO("mysql:host=localhost", "root", "root");
```

. Se utilizan las palabras reservadas `pgsql`, `sqlite`, `firebird`, `informix` y `OCI` para establecer conexión con PostgreSQL, SQLite, Firebird, Informix y Oracle respectivamente.

Listado completo de una tabla

La extracción de datos de una tabla mediante la interfaz PDO se realiza de forma casi idéntica a como se hace mediante `mysqli`. Los datos de cada fila se transforman en objetos mediante `fetchObject()`.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <h2>
      Base de datos <u>banco</u><br>
      Tabla <u>cliente</u><br>
    </h2>
    <?php

      // Conexión a la base de datos
      try {
        $conexion = new PDO("mysql:host=localhost;dbname=banco;charset=utf8", "root", "root");
      } catch (PDOException $e) {
        echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
        die ("Error: " . $e->getMessage());
      }

      $consulta = $conexion->query("SELECT dni, nombre, direccion, telefono FROM cliente");

      <?>
      <table border="1">
        <tr>
          <td><b>DNI</b></td>
          <td><b>Nombre</b></td>
          <td><b>Dirección</b></td>
          <td><b>Teléfono</b></td>
        </tr>

        <?php
        while ($cliente = $consulta->fetchObject()) {
          <?>
          <tr>
            <td><?= $cliente->dni ?></td>
            <td><?= $cliente->nombre ?></td>
            <td><?= $cliente->direccion ?></td>
            <td><?= $cliente->telefono ?></td>
          </tr>
        <?php
      }
    <?>
  </table>

```

```

    <br>
    Número de clientes: <?= $consulta->rowCount() ?>

    <?php $conexion->close(); ?>
</body>
</html>

```

8.5 Operaciones sobre una tabla

Las operaciones de inserción, borrado o actualización de datos sobre una tabla se realiza mediante `exec()`.

Como ejemplo, vamos a realizar una inserción mediante el comando `INSERT` en la tabla `cliente` de la base de datos `banco`.

En primer lugar se piden los datos del cliente mediante un formulario como el que se muestra a continuación.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <h1>Base de datos <u>banco</u></h1>
    <h2>Alta de cliente</h2>
    <form action="altaClienteAccion.php" method="post">
      DNI <input type="text" name="dni" required><br>
      Nombre <input type="text" name="nombre"><br>
      Dirección <input type="text" name="direccion"><br>
      Teléfono <input type="tel" name="telefono"><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>

```

Los datos recogidos en el formulario se envían a `altaClienteAccion.php`.

Antes de realizar la inserción del registro en la base de datos es necesario comprobar que no existe ningún cliente con el DNI del nuevo cliente que se quiere insertar. Una vez comprobado este supuesto, se procede a ejecutar el `INSERT` con los datos del nuevo cliente.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      // Conexión a la base de datos
      try {
        $conexion = new PDO("mysql:host=localhost;dbname=banco;charset=utf8", "root", "root");
      } catch (PDOException $e) {
        echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
        die ("Error: " . $e->getMessage());
      }

      // Comprueba si ya existe un cliente con el DNI introducido
      $consulta = $conexion->query("SELECT dni FROM cliente WHERE dni=".$_POST['dni']);

      if ($consulta->rowCount() > 0) {
        ?>
        Ya existe un cliente con DNI <?= $_POST['dni'] ?><br>
        Por favor, vuelva a la <a href="altaClienteFormulario.php">página de alta de cliente</a>.
      } else {
        $insercion = "INSERT INTO cliente (dni, nombre, direccion, telefono) VALUES ('".$_POST['dni'],
        $_POST['nombre'], $_POST['direccion'], $_POST['telefono'])";
        //echo $insercion;
        $conexion->exec($insercion);
        echo "Cliente dado de alta correctamente.";
        $conexion->close();
      }
    ?>
  </body>
</html>

```



















8.6 Ejercicios



Ejercicio 1

Crea una aplicación web que permita hacer listado, alta, baja y modificación sobre la tabla `cliente` de la base de datos `banco`.

- Para realizar el listado bastará un `SELECT`, tal y como se ha visto en los ejemplos.
- El alta se realizará mediante un formulario donde se especificarán todos los campos del nuevo registro. Luego esos datos se enviarán a una página que ejecutará `INSERT`.
- Para realizar una baja, se mostrará un botón que ejecutará `DELETE`.
- La modificación se realiza mediante `UPDATE`.

DNI	Nombre	Dirección	Teléfono		
3534534	Cacerolo Tontoñez	Almogía	123456	 Eliminar	 Modificar
45678	Mota	Calle Falsa, 123	555 444333	 Eliminar	 Modificar
456958	Javier Roviralta	Calle Olvido, 77	555 76845	 Eliminar	 Modificar
555	Luis José	Montserrat Roig, 10	5555 234233	 Eliminar	 Modificar
657456	uytutyut	ghjfhgj	67867	 Eliminar	 Modificar
65767	Pepito Lupiañez	Alhaurin	867867867	 Eliminar	 Modificar
76859	Ignacio	Periquito, 333	555 325476	 Eliminar	 Modificar
789654	Yren	Calle Verdadera, 98	555 98765	 Eliminar	 Modificar
873475933	Maria Sol	Calle Flor	555 123456	 Eliminar	 Modificar
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	 Nuevo cliente	



Ejercicio 2

Modifica el programa anterior añadiendo las siguientes mejoras:

- El listado debe aparecer paginado en caso de que haya muchos clientes.
- Al hacer un alta, se debe comprobar que no exista ningún cliente con el DNI introducido en el formulario.
- La opción de borrado debe pedir confirmación.
- Cuando se realice la modificación de los datos de un cliente, los campos que no se han cambiado deberán permanecer inalterados en la base de datos.



Ejercicio 3

Modifica la tienda virtual realizada anteriormente de tal forma que todos los datos de los artículos se guarden en una base de datos.



Ejercicio 4

Crea el programa GESTISIMAL (GESTIÓN SIMPlificada de Almacén) para llevar el control de los artículos de un almacén. De cada artículo se debe saber el código, la descripción, el precio de compra, el precio de venta y el stock (número de unidades). La entrada y salida de mercancía supone respectivamente el incremento y decremento de stock de un determinado artículo. Hay que controlar que no se pueda sacar más mercancía de la que hay en el almacén. El programa debe tener, al menos, las siguientes funcionalidades: listado, alta, baja, modificación, entrada de mercancía y salida de mercancía.

GESTISIMAL

Código	Descripción	Precio de compra	Precio de venta	Margen	Stock				
h020	Barra acero 16mm. longitud 6m.	35.30	45.40	10.1	50	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>
h007	barra para cortina 2,00 m.	10.30	22.33	12.03	5	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>
h005	Caja tuercas 16mm.	21.00	25.05	4.05	20	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>
h006	chapa galvanizada	10.50	20.55	10.05	3	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>
m001	Estanteria para pared.	25.30	30.60	5.3	5	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>

Página 1 de 3

Primera

Anterior

Siguiente

Última

Código	Descripción	Precio de compra	Precio de venta	Stock	
<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div>Nuevo artículo</div>



Ejercicio 5

Modifica el programa anterior añadiendo las siguientes mejoras:

- Comprueba la existencia del código en el alta, la baja y la modificación de artículos para evitar errores.
- Cambia la opción “Salida de stock” por “Venta”. Esta nueva opción permitirá hacer una venta de varios artículos y emitir la factura correspondiente. Se debe preguntar por los códigos y las cantidades de cada artículo que se quiere comprar. Aplica un 21% de IVA.

9. PHP orientado a objetos

9.1 El paradigma de la Programación Orientada a Objetos

Mientras que la programación estructurada se basa en la utilización de estructuras de control como las sentencias `if` y los bucles `for` y `while`; y usa como almacenamiento de información las variables y los arrays; la programación orientada a objetos se basa, como su nombre indica, en la utilización de objetos.

La programación estructurada y la POO no son excluyentes, un programa basado en objetos seguirá teniendo variables, bucles y sentencias condicionales, no obstante éste último estará seguramente mejor organizado y será más legible y escalable.

Un **objeto** en términos de POO no se diferencia mucho de lo que conocemos como un objeto en la vida real. Pensemos por ejemplo en un coche. Nuestro coche sería un objeto concreto de la vida real, igual que el coche del vecino, o el coche de un compañero de trabajo, o un deportivo que vimos por la calle el fin de semana pasado... Todos esos coches serían objetos concretos que podemos ver y tocar. Usando la terminología de la POO diríamos que son instancias.

Tanto mi coche como el coche del vecino tienen algo en común, ambos son coches. En este caso mi coche y el coche del vecino serían **instancias** (objetos) y coche (a secas) sería una **clase**. La palabra coche define algo genérico, es una abstracción, no es un coche concreto sino que hace referencia a algo que tiene una serie de propiedades como matrícula, marca, modelo, color, etc. Este conjunto de propiedades se denominan **atributos** o **variables de instancia**.



Clase

Concepto abstracto que denota una serie de cualidades, por ejemplo **coche**.

Instancia

Objeto palpable, que se deriva de la concreción de una clase, por ejemplo **mi coche**.

Atributos

Conjunto de características que comparten los objetos de una clase, por ejemplo para la clase **coche** tendríamos **matrícula**, **marca**, **modelo**, **color** y **número de plazas**.

9.2 Encapsulamiento y ocultación

Uno de los pilares en los que se basa la Programación Orientada a Objetos es el **encapsulamiento**. Básicamente, el **encapsulamiento** consiste en definir todas las propiedades y el comportamiento de una clase dentro de esa clase; es decir, en la clase `Coche` estará definido todo lo concerniente a la clase `Coche` y en la clase `Libro` estará definido todo lo que tenga que ver con la clase `Libro`.

El **encapsulamiento** parece algo obvio, casi de perogrullo, pero hay que tenerlo siempre muy presente al programar utilizando clases y objetos. En alguna ocasión puede que estemos tentados a mezclar parte de una

clase con otra clase distinta para resolver un problema puntual. No hay que caer en esa trampa. Se deben escribir los programas de forma que cada cosa esté en su sitio. Sobre todo al principio, cuando definimos nuestras primeras clases, debemos estar pendientes de que todo está definido donde corresponde.

La **ocultación** es una técnica que incorporan algunos lenguajes (entre ellos Java) que permite esconder los elementos que definen una clase, de tal forma que desde otra clase distinta no se pueden “ver las tripas” de la primera. La **ocultación** facilita, como veremos más adelante, el **encapsulamiento**.

9.3 Implementación de clases en PHP

Las clases en PHP comienzan con una letra mayúscula. Es muy recomendable separar la implementación de las clases del programa principal en ficheros diferentes. Desde el programa principal se puede cargar la clase mediante `include` o `include_once` seguido del nombre del fichero de clase. El nombre de la clase debe coincidir con el nombre del fichero que la implementa (con la extensión `.php`).

A continuación tenemos un ejemplo muy sencillo. Se trata de la implementación de la clase `Persona`. Esta clase tendrá dos atributos: `nombre` y `profesión`.

Se define también el **constructor**. Este método es muy importante ya que se llamará siempre que se creen nuevos objetos de la clase y servirá generalmente para inicializar los valores de los atributos. En PHP, el constructor de una clase se define con el nombre `__construct()`

Se crea además un método que, aplicado a una instancia de la clase `Persona`, muestra un mensaje por pantalla.

```
<?php
class Persona {
    private $nombre;
    private $profesion;

    // Constructor
    public function __construct($nom, $pro) {
        $this->nombre = $nom;
        $this->profesion = $pro;
    }

    public function presentarse() {
        echo "Hola, me llamo " . $this->nombre . " y soy " . $this->profesion . "<br>";
    }
}
```

Observa que los atributos se declaran privados y los métodos se declaran públicos. Eso quiere decir que los atributos serán accesibles únicamente desde la implementación de la clase y que los métodos se podrán utilizar en cualquier lugar siempre que se apliquen a los objetos de la clase adecuada, en este caso a las instancias de `Persona`. Salvo que se indique lo contrario, seguiremos esta regla al realizar los ejercicios.

En el programa principal `index.php` se carga el fichero con la implementación de la clase, se crean dos instancias de la clase `Persona` y se les aplica el método `presentarse`. Mediante la función `var_dump()` se puede visualizar el tipo y el valor de todos los atributos que tiene un objeto concreto.


```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      include_once 'Persona.php';

      $unTipo = new Persona("Pepe Pérez", "albañil");
      $unNota = new Persona("Rigoberto Peláez", "programador");
      $unTipo->presentarse();
      $unNota->presentarse();
      var_dump($unNota);
      var_dump($unTipo);
    ?>
  </body>
</html>

```

Vamos a mejorar un poco nuestra clase Persona. Es poco elegante y sobre todo poco práctico hacer un echo desde un método. Es mejor devolver una cadena de caracteres para después procesarla en el programa principal aplicándole estilos, guardándola en una variable, etc.

```

<?php
class Persona {
  private $nombre;
  private $profesion;

  // Constructor
  public function __construct($nom, $pro) {
    $this->nombre = $nom;
    $this->profesion = $pro;
  }

  public function presentarse() {
    return "Hola, me llamo " . $this->nombre . " y soy " . $this->profesion . "<br>";
  }
}

```



Recuerda que los ficheros que contienen únicamente código PHP comienzan con la etiqueta <?php pero no es necesario que terminen con ?>. Es más, se recomienda no usar la etiqueta de cierre.

Ahora, el programa que utiliza la clase Persona sería ligeramente diferente.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      include_once 'Persona.php';
      include_once 'Perro.php';

      $unTipo = new Persona("Pepe Pérez", "albañil");
      $unNota = new Persona("Rigoberto Peláez", "programador");
      echo $unTipo->presentarse();
      echo $unNota->presentarse();
    ?>
  </body>
</html>

```

El constructor es un método especial y por ello comienza con doble carácter de subrayado. Hay otro método especial muy útil, se trata de `__toString`. Si se define el método `__toString` en una clase, cuando se realice un `echo` sobre un elemento de esa clase, se ejecutará dicho `__toString`.

En la siguiente versión de la clase `Persona` se muestra la implementación del método `__toString`.

```

<?php
class Persona {
  private $nombre;
  private $profesion;
  private $edad;

  // Constructor
  public function __construct($nom, $pro, $edad) {
    $this->nombre = $nom;
    $this->profesion = $pro;
    $this->edad = $edad;
  }

  public function presentarse() {
    return "Hola, me llamo " . $this->nombre . " y soy " . $this->profesion . "<br>";
  }

  public function __toString() {
    return "<hr><b>$this->nombre</b><br>
      Profesión: $this->profesion<br>
      Edad: $this->edad<hr>";
  }
}

```

```
}  
}
```

En el programa principal `-index.php-` se ejecuta el método `__toString` haciendo `echo` sobre el objeto de la clase `Persona`.

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="UTF-8">  
    <title></title>  
  </head>  
  <body>  
    <?php  
      include_once 'Persona.php';  
  
      $unTipo = new Persona("Pepe Pérez", "albañil", 30);  
      $unNota = new Persona("Rigoberto Peláez", "programador");  
      echo $unTipo;  
      echo $unNota;  
    ?>  
  </body>  
</html>
```

Cuando se implementa una clase en PHP, igual que se hace con otros lenguajes - es habitual crear métodos *getter* y *setter*. Se trata de métodos muy simples. El cometido de un *getter* es proporcionar (devolver) el valor de un atributo mientras que la misión de un *setter* es darle un valor a un atributo; este valor que se quiere asignar se pasa como parámetro.



Un elemento `public` (público) es visible desde cualquier clase, un elemento `protected` (protegido) es visible desde la clase actual y desde todas sus subclases (en el siguiente apartado se estudia el concepto de subclase) y, finalmente, un elemento `private` (privado) únicamente es visible dentro de la clase actual. Por regla general, se suelen definir los atributos como privados y los métodos como públicos.

A continuación se muestra la implementación de la clase `Gato`. El método `getSexo()` es un *getter* que devuelve el valor del atributo `$sexo` (macho o hembra).

<?php

```
class Gato {

    // atributos

    private $color;
    private $raza;
    private $edad;
    private $peso;
    private $sexo;

    // métodos

    public function __construct($s) {
        $this->sexo = $s;
    }

    public function getSexo() {
        return $this->sexo;
    }

    public function maulla() {
        echo "Miauuuu<br>";
    }

    public function ronronea() {
        echo "mrrrrrrr<br>";
    }

    public function come($comida) {
        if ($comida == "pescado") {
            echo "Hmmm, gracias<br>";
        } else {
            echo "Lo siento, yo solo como pescado<br>";
        }
    }

    public function peleaCon($contrincante) {
        if (($this->getSexo()) == "hembra") {
            echo "no me gusta pelear<br>";
        } else if (($contrincante->getSexo()) == "hembra") {
            echo "no peleo contra gatitas<br>";
        } else {
```

```
        echo "ven aquí que te vas a enterar<br>";
    }
}
}
```

El programa que prueba la clase Gato es el siguiente.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      include_once 'Gato.php';

      $garfield = new Gato("macho");

      echo "hola gatito<br>";
      $garfield->maulla();

      echo "toma tarta<br>";
      $garfield->come("tarta selva negra");
      echo "toma pescado, a ver si esto te gusta<br>";
      $garfield->come("pescado");

      $tom = new Gato("macho");

      echo "Tom, toma sopita de verduras<br>";
      $tom->come("sopa de verduras");

      $lisa = new Gato("hembra");

      echo "gatitos, a ver cómo maulláis<br>";
      $garfield->maulla();
      $tom->maulla();
      $lisa->maulla();

      $garfield->peleaCon($lisa);
      $lisa->peleaCon($tom);
      $tom->peleaCon($garfield);
    ?>
  </body>
</html>
```

9.4 Herencia

La herencia es una de las características más importantes de la POO. Si definimos una serie de atributos y métodos para una clase, al crear una subclase, todos estos atributos y métodos siguen siendo válidos.

A continuación se muestra la implementación de la clase `Animal`. Uno de los métodos de esta clase es `duerme`. Luego crearemos las clases `Gato` y `Ave` como subclases de `Animal`. De forma automática, se podrá utilizar el método `duerme` con las instancias de las clases `Gato` y `Ave` ¿no es fantástico?

```
<?php
```

```
abstract class Animal {

    private $sexo;

    public function __construct($s = "macho") {
        $this->sexo = $s;
    }

    public function __toString() {
        return "Sexo: $this->sexo";
    }

    public function getSexo() {
        return $this->sexo;
    }

    public function duerme() {
        return "Zzzzzzz";
    }

    public function aseate() {
        return "Me gusta asearme, soy un animal.<br>";
    }
}
```

Observa que la definición de la clase `Animal` comienza con la siguiente línea.

```
abstract class Animal {
```

Por tanto, `Animal` será una clase abstracta.



Clase abstracta (abstract)

Una clase abstracta es aquella que no va a tener instancias de forma directa, aunque sí habrá instancias de las subclases (siempre que esas subclases no sean también abstractas). Por ejemplo, si se define la clase `Animal` como abstracta, no se podrán crear objetos de la clase `Animal`, es decir, no se podrá hacer `$mascota = new Animal()`, pero sí se podrán crear instancias de la clase `Gato`, `Ave` o `Pinguino` que son subclases de `Animal`.

La clase `Ave` es subclase de `Animal` y la clase `Pinguino`, a su vez, sería subclase de `Ave` y por tanto hereda todos sus atributos y métodos.

Para crear en PHP una subclase de otra clase existente se utiliza la palabra reservada `extends`. A continuación se muestra el código de las clases `Gato`, `Ave` y `Pinguino`, así como el programa que prueba estas clases creando instancias y aplicándoles métodos.

```
<?php
```

```
include_once 'Animal.php';

class Gato extends Animal {

    private $raza;

    public function __construct($s, $r) {
        parent::__construct($s);
        if (isset($r)) {
            $this->raza = $r;
        } else {
            $this->raza = "siamés";
        }
    }

    public function __toString() {
        return parent::__toString() . "<br>Raza: $this->raza";
    }

    public function maulla() {
        return "Miauuuu<br>";
    }

    public function ronronea() {
        return "mrrrrrr<br>";
    }

    public function come($comida) {
        if ($comida == "pescado") {
```

```

        return "Hmmm, gracias<br>";
    } else {
        return "Lo siento, yo solo como pescado<br>";
    }
}

public function peleaCon($contrincante) {
    if (($this->getSexo()) == "hembra") {
        echo "no me gusta pelear<br>";
    } else if (($contrincante->getSexo()) == "hembra") {
        echo "no peleo contra gatitas<br>";
    } else {
        echo "ven aquí que te vas a enterar<br>";
    }
}
}
}

```

Observa cómo se indica que la clase Gato es subclase de Animal.

```
class Gato extends Animal
```

Para que el programa reconozca que existe la clase Animal es necesario incluirla en el archivo actual.

```
include_once 'Animal.php';
```

Mediante parent se puede hacer una llamada al método de la clase padre. Por ejemplo parent::__construct(\$s); dentro de la definición de la clase Gato invoca al constructor de la clase padre, es decir, la clase Animal.

A continuación tenemos la definición de la clase Ave que es una subclase de Animal.

```
<?php
```

```

include_once 'Animal.php';

class Ave extends Animal {

    public function __construct($s) {
        parent::__construct($s);
    }

    public function aseate() {
        return "Me estoy limpiando las plumas<br>" . parent::aseate();
    }
}

```



```

    public function vuela() {
        return "Estoy volando<br>";
    }
}

```

El siguiente código corresponde a la definición de la clase Pinguino que es subclase de Ave y, por lo tanto, también es subclase de Animal. Observa cómo se sobrescribe el método `vuela()`. Cuando se define un método en una subclase con el mismo nombre que tiene en la superclase, tiene preferencia en la ejecución el método de la subclase.

```
<?php
```

```

include_once 'Ave.php';

class Pinguino extends Ave {

    public function __construct($s) {
        parent::__construct($s);
    }

    public function aseate() {
        return parent::aseate() . "A los pingüinos nos gusta asearnos<br>";
    }

    public function vuela() {
        return "No puedo volar<br>";
    }
}

```

Observa que no es necesario incluir el fichero `Animal.php` en la definición de la clase Pinguino ya que está incluido en la definición de la clase Ave.

Por último mostramos el programa que prueba las clases definidas anteriormente.

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
            include_once 'Animal.php'; // no es necesario incluirla
            include_once 'Ave.php'; // no es necesario incluirla

```

```

include_once 'Pinguino.php';
include_once 'Gato.php';

$garfield = new Gato("macho", "romano");
$tom = new Gato("macho");
$lisa = new Gato("hembra");
$silvestre = new Gato();

echo "$garfield

---

";
echo "$tom

---

";
echo "$lisa

---

";
echo "$silvestre

---

";

$miLoro = new Ave();
echo $miLoro->aseate();
echo $miLoro->vuela();

$pingu = new Pinguino("hembra");
echo $pingu->aseate();
echo $pingu->vuela();
?>
</body>
</html>

```

No es necesario incluir las siguientes líneas:

```

include_once 'Animal.php';
include_once 'Ave.php';

```

Ya que los ficheros `Animal.php` y `Ave.php` se cargan desde `Gato.php` y `Pinguino.php` respectivamente. No obstante, se pueden dejar por claridad, para indicar todas las clases que intervienen en el programa.

9.5 Atributos y métodos de clase (static)

Hasta el momento hemos definido atributos de instancia como `$raza` o `$sexo` y métodos de instancia como `maulla()`, `come()` o `vuela()`. De tal modo que si en el programa se crean 20 gatos, cada uno de ellos tiene su propia raza y puede haber potencialmente 20 razas diferentes. También podría aplicar el método `maulla()` a todos y cada uno de esos 20 gatos.

No obstante, en determinadas ocasiones, nos puede interesar tener atributos de clase (variables de clase) y métodos de clase. Cuando se define una variable de clase solo existe una copia del atributo para toda la clase y no una para cada objeto. Esto es útil cuando se quiere llevar la cuenta global de algún parámetro. Los métodos de clase se aplican a la clase y no a instancias concretas.

A continuación se muestra un ejemplo que contiene la variable de clase `$kilometrajeTotal`. Si bien cada coche tiene un atributo `$kilometraje` donde se van acumulando los kilómetros que va recorriendo, en la

variable de clase `$kilometrajeTotal` se lleva la cuenta de los kilómetros que han recorrido todos los coches que se han creado.

También se crea un método de clase llamado `getKilometrajeTotal()` que simplemente es un *getter* para la variable de clase `$kilometrajeTotal`.

```
<?php
```

```
class Coche {

    // atributo de clase
    private static $kilometrajeTotal = 0;

    // método de clase
    public static function getKilometrajeTotal() {
        return Coche::$kilometrajeTotal;
    }

    private $marca;
    private $modelo;
    private $kilometraje;

    public function __construct($ma, $mo) {
        $this->marca = $ma;
        $this->modelo = $mo;
        $this->kilometraje = 0;
    }

    public function getKilometraje() {
        return $this->kilometraje;
    }

    public function recorre($km) {
        $this->kilometraje += $km;
        Coche::$kilometrajeTotal += $km;
    }
}
```

El atributo `$kilometrajeTotal` almacena el número total de kilómetros que recorren todos los objetos de la clase `Coche`, es un único valor, por eso se declara como `static`. Por el contrario, el atributo `$kilometraje` almacena los kilómetros recorridos por un objeto concreto y tendrá un valor distinto para cada uno de ellos. Si en el programa principal se crean 20 objetos de la clase `Coche`, cada uno tendrá su propio `$kilometraje`.

A continuación se muestra el programa que prueba la clase `Coche`.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      include_once 'Coche.php';

      $cocheDeLuis = new Coche("Saab", "93");
      $cocheDeJuanK = new Coche("Toyota", "Avensis");

      $cocheDeLuis->recorre(30);
      $cocheDeLuis->recorre(40);
      $cocheDeLuis->recorre(220);
      $cocheDeJuanK->recorre(60);
      $cocheDeJuanK->recorre(150);
      $cocheDeJuanK->recorre(90);
      echo "El coche de Luis ha recorrido " . $cocheDeLuis->getKilometraje() . "Km<br>";
      echo "El coche de Juan Carlos ha recorrido " . $cocheDeJuanK->getKilometraje() . "Km\
<br>";
      echo "El kilometraje total ha sido de " . Coche::getKilometrajeTotal() . "Km";
    ?>
  </body>
</html>

```

El método `getKilometrajeTotal()` se aplica a la clase `Coche` por tratarse de un método de clase (método `static`). Este método no se podría aplicar a una instancia, de la misma manera que un método que no sea `static` no se puede aplicar a la clase sino a los objetos.

9.6 Serialización de objetos

Como hemos visto en capítulos anteriores, cuando se recarga una página se pierden todos los datos que no se hayan guardado en una sesión. Lo mismo sucede con los objetos, si se crean instancias de una clase, se perderán en el momento en que se recargue la página o cuando el flujo de la aplicación nos lleve a una página diferente a la actual.

Los objetos creados a partir de una clase se pueden guardar en variables de sesión pero es necesario tratarlos convenientemente; hay que serializarlos antes de asignarlos. En el proceso de serialización, una estructura - en este caso un objeto - queda transformada en una cadena de caracteres. Cuando se quiera recuperar el objeto a partir de la sesión, habrá que hacer el proceso inverso, es decir des-serializar la variable de sesión para obtener la instancia.

Aunque parece algo complicado, este proceso se verá claramente con un ejemplo.

En primer lugar definimos la clase `MonstruoDeLasGalletas`.

```
<?php
```

```
class MonstruoDeLasGalletas {

    private $galletas; // galletas comidas

    public function __construct($s) {
        $this->galletas = 0;
    }

    public function getGalletas() {
        return $this->galletas;
    }

    public function come($g) {
        $this->galletas = $this->galletas + $g;
    }
}
```

El programa principal es el siguiente.

```
<?php
```

```
session_start();

include_once 'MonstruoDeLasGalletas.php';

if (!isset($_SESSION['coco'])) {
    $_SESSION['coco'] = serialize(new MonstruoDeLasGalletas());
}

$coco = unserialize($_SESSION['coco']);
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php

        $numeroDeGalletas = $_POST['numeroDeGalletas'];
        if (isset($numeroDeGalletas)) {
            $coco->come($numeroDeGalletas);
```

```

    }
    ?>

    <h2>Soy Coco y he comido <?=$coco->getGalletas(); ?> galletas.</h2>
    <form action="index.php" method="POST">
        Nº de galletas:
        <input type="number" name="numeroDeGalletas" min="1">
        <input type="submit" value="Comer">
    </form>

    <?php
        $_SESSION['coco'] = serialize($coco);
    ?>

</body>
</html>

```

En la primera carga de página se crea un objeto de la clase `MonstruoDeLasGalletas`, se serializa ese objeto y se guarda en una variable de sesión con nombre `$_SESSION['coco']`.

```

if (!isset($_SESSION['coco'])) {
    $_SESSION['coco'] = serialize(new MonstruoDeLasGalletas());
}

```

Antes de operar con el objeto, debe extraerse de la sesión.

```

$coco = unserialize($_SESSION['coco']);

```

A partir de ahora, `$coco` es un objeto de la clase `MonstruoDeLasGalletas` al que le podemos aplicar cualquiera de los métodos definidos.

Con el fin de conservar el objeto en memoria, lo serializamos y lo guardamos en la sesión.

```

$_SESSION['coco'] = serialize($coco);

```

9.7 Ejercicios



Ejercicio 1

Crea las clases `Animal`, `Mamifero`, `Ave`, `Gato`, `Perro`, `Canario`, `Pinguino` y `Lagarto`. Crea, al menos, tres métodos específicos de cada clase y redefine el/los método/s cuando sea necesario. Prueba las clases en un programa en el que se instancien objetos y se les apliquen métodos. Puedes aprovechar las capacidades que proporciona HTML y CSS para incluir imágenes, sonidos, animaciones, etc. para representar acciones de objetos; por ejemplo, si el canario canta, el perro ladra, o el ave vuela.



Ejercicio 2

Crea la clase `Vehiculo`, así como las clases `Bicicleta` y `Coche` como subclases de la primera. Para la clase `Vehiculo`, crea los métodos de clase `getVehiculosCreados()` y `getKmTotales()`; así como el método de instancia `getKmRecorridos()`. Crea también algún método específico para cada una de las subclases. Prueba las clases creadas mediante una aplicación que realice, al menos, las siguientes acciones:

- Anda con la bicicleta
- Haz el caballito con la bicicleta
- Anda con el coche
- Quema rueda con el coche
- Ver kilometraje de la bicicleta
- Ver kilometraje del coche
- Ver kilometraje total



Ejercicio 3

Crea la clase `DadoPoker`. Las caras de un dado de poker tienen las siguientes figuras: As, K, Q, J, 7 y 8. Crea el método `tira()` que no hace otra cosa que tirar el dado, es decir, genera un valor aleatorio para el objeto al que se le aplica el método. Crea también el método `nombreFigura()`, que diga cuál es la figura que ha salido en la última tirada del dado en cuestión. Crea, por último, el método `getTiradasTotales()` que debe mostrar el número total de tiradas entre todos los dados. Realiza una aplicación que permita tirar un cubilete con cinco dados de poker.



Ejercicio 4

Queremos gestionar la venta de entradas (no numeradas) de **Expocoche** **Campanillas** que tiene 3 zonas, la sala principal con 1000 entradas disponibles, la zona de compra-venta con 200 entradas disponibles y la zona vip con 25 entradas disponibles. Hay que controlar que existen entradas antes de venderlas. Define la clase `Zona` con sus atributos y métodos correspondientes y crea un programa que permita vender las entradas. En la pantalla principal debe aparecer información sobre las entradas disponibles y un formulario para vender entradas. Debemos indicar para qué zona queremos las entradas y la cantidad de ellas. Lógicamente, el programa debe controlar que no se puedan vender más entradas de la cuenta.

10. Modelo Vista Controlador

El Modelo Vista Controlador (Model View Controller) - también conocido por sus siglas MVC - es un patrón de diseño de software que separa una aplicación en tres grandes bloques: el acceso a datos, la interfaz de usuario y la lógica de negocio.

La manera de implementar este patrón no es única. En este libro mostramos una de las muchas aproximaciones posibles, pero el lector podrá encontrar implementaciones diferentes en otros manuales.

Para ilustrar los conceptos del patrón MVC vamos a trabajar con un ejemplo muy simple: un gestor de ofertas de una pizzería. Los datos sobre las ofertas se guardan en una tabla llamada `oferta` dentro de la base de datos `pizzeria`. El fichero `pizzeria.sql` se encuentra disponible en [GitHub](#)¹. Cada oferta tiene un identificador único que es un número entero. El campo correspondiente en la base de datos es un entero “auto-incrementable”, por tanto, el propio **MySQL** se encarga de asignar un valor cuando se crea una nueva oferta. Para cada oferta hay también un título, una imagen (el nombre del fichero de imagen) y una descripción.

La estructura de la aplicación quedaría de la siguiente manera:

```
.
├── Controller
│   ├── borraOferta.php
│   ├── grabaOferta.php
│   ├── index.php
│   └── nuevaOferta.php
├── index.php
├── Model
│   ├── Oferta.php
│   └── PizzeriaDB.php
└── View
    ├── formularioOferta.php
    ├── images
    │   ├── pizza1.jpg
    │   ├── pizza2.jpg
    │   └── pizza3.jpg
    └── listado.php
```

10.1 El modelo

En una aplicación web, los datos se guardan normalmente en una base de datos. La parte del modelo dentro del MVC será, por tanto, una capa que da acceso a los datos y que abstrae mediante una clase todas las consultas, inserciones, borrados, etc. que se realizan en las tablas mediante SQL.

¹<https://github.com/LuisJoseSanchez/aprende-php-con-ejercicios>

Por ejemplo, si una aplicación gestiona artículos de un almacén, deberá tener una tabla - con nombre `articulo` - en la base de datos que almacene los datos de los artículos: código, descripción, tipo, precio, etc. En este caso, como parte del modelo habrá una clase con nombre `Articulo.php` donde estará implementado el acceso a los datos de los artículos para sacar listados, dar de alta nuevos artículos, modificar artículos existentes, etc.

Un ejemplo de oferta podría ser la que tiene como identificador el 3, como título *“Bebida gratis pidiendo dos pizzas”*, como imagen *“pizza3.jpg”* y como descripción algún texto más largo como *“Pidiendo dos pizzas de cualquier tipo te regalamos dos bebidas (no incluye bebidas alcohólicas de alta graduación)”*.

El modelo estará implementado dentro de la carpeta `Model`. Esta carpeta contendrá todo el código que tenga que ver con el acceso a los datos.

El fichero que se encarga del establecimiento de la conexión con la base de datos es `PizzeriaDB.php` y se muestra a continuación.

```
<?php
```

```
abstract class PizzeriaDB {
    private static $server = 'localhost';
    private static $db = 'pizzeria';
    private static $user = 'root';
    private static $password = 'root';

    public static function connectDB() {
        try {
            $connection = new PDO("mysql:host=".self::$server.";dbname=".self::$db.";charset=utf\
8", self::$user, self::$password);
        } catch (PDOException $e) {
            echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
            die ("Error: " . $e->getMessage());
        }

        return $connection;
    }
}
```

A partir de este momento, cada vez que queramos establecer una conexión con la base de datos bastará con escribir `$miConexion = PizzeriaDB::connectDB()`.

Si en el futuro se cambia el nombre de usuario de la base de datos, la contraseña o algún otro dato de la conexión, únicamente habría que actualizar el fichero `PizzeriaDB.php`; el resto de la aplicación permanecería intacta.

A continuación, definiremos la clase `Oferta` en base a la tabla `oferta` de la base de datos. Los atributos de instancia deben tener los mismos nombres que los campos de la tabla.

En la clase `Oferta` se define el constructor y los *setter*.

El método `insert()` graba los datos de un objeto de la clase `Oferta` en la base de datos. El método `delete()` aplicado a un objeto, mira el identificador de la oferta que tiene ese objeto y borra en la base de datos el registro que tiene ese identificador.

El método `getOfertas()` devuelve un array de objetos con toda la información de las ofertas disponibles.

Por último, el método `getOfertaById()` devuelve un objeto con la información de la oferta que tiene un determinado identificador. Obviamente, los datos de esa oferta se han tenido que buscar en la base de datos.

El código de `Oferta.php` se muestra a continuación.

```
<?php
```

```
require_once 'PizzeriaDB.php';
```

```
class Oferta {
    private $id;
    private $titulo;
    private $imagen;
    private $descripcion;

    function __construct($id, $titulo, $imagen, $descripcion) {
        $this->id = $id;
        $this->titulo = $titulo;
        $this->imagen = $imagen;
        $this->descripcion = $descripcion;
    }

    public function getId() {
        return $this->id;
    }

    public function getTitulo() {
        return $this->titulo;
    }

    public function getImagen() {
        return $this->imagen;
    }

    public function getDescripcion() {
        return $this->descripcion;
    }

    public function insert() {
        $conexion = PizzeriaDB::connectDB();
        $insercion = "INSERT INTO oferta (titulo, imagen, descripcion) VALUES ('".$this->titu\
```

```

lo."\\", \\\"".$this->imagen."\\", \\\"".$this->descripcion."\\");
    $conexion->exec($insercion);
}

public function delete() {
    $conexion = PizzeriaDB::connectDB();
    $borrado = "DELETE FROM oferta WHERE id=\\\"".$this->id."\\\"";
    $conexion->exec($borrado);
}

public static function getOfertas() {
    $conexion = PizzeriaDB::connectDB();
    $seleccion = "SELECT id, titulo, imagen, descripcion FROM oferta";
    $consulta = $conexion->query($seleccion);

    $ofertas = [];

    while ($registro = $consulta->fetchObject()) {
        $ofertas[] = new Oferta($registro->id, $registro->titulo, $registro->imagen, $regist\
ro->descripcion);
    }

    return $ofertas;
}

public static function getOfertaById($id) {
    $conexion = PizzeriaDB::connectDB();
    $seleccion = "SELECT id, titulo, imagen, descripcion FROM oferta WHERE id=\\\"".$id."\\\"";
    $consulta = $conexion->query($seleccion);
    $registro = $consulta->fetchObject();
    $oferta = new Oferta($registro->id, $registro->titulo, $registro->imagen, $registro->d\
escripcion);

    return $oferta;
}
}

```

El objetivo de crear `Oferta.php` no es otro que abstraer el acceso a la base de datos. A partir de este momento ya no tendremos que lidiar directamente con las conexiones, los `SELECT`, `INSERT`, etc. ya que eso lo hace ahora la clase `Oferta`.

10.2 La vista

En una aplicación realizada mediante el patrón MVC, la vista corresponde a la interfaz de usuario. La vista tiene código HTML y unos “huecos” donde se colocará la información que pasará el controlador. Normalmente

la vista - al ser la interfaz de usuario - contiene CSS, Javascript, JQuery, etc.

Todo lo que tenga que ver con la vista se coloca en la carpeta `View`. Normalmente las imágenes y otros ficheros que necesita la aplicación se colocan aparte, en una carpeta con nombre `Assets` (activos). Nosotros para simplificar colocaremos las imágenes dentro de la vista, dentro de la carpeta `images`.

Los datos que le pasa el controlador a la vista van en un array asociativo llamado `$data[]`. Si el controlador y la vista lo hacen personas diferentes, se tendrán que poner de acuerdo en los nombres de los datos que se pasan de un sitio a otro. Por ejemplo, el programador que implementa el controlador le puede decir al diseñador que se encarga de la interfaz que el nombre del usuario logueado es `$data['usuario']`; de esta forma, el diseñador podría colocar en la interfaz el nombre de usuario escribiendo por ejemplo:

```
<h3>Usuario: <?= $data['usuario'] ?></h3>
```

A continuación se muestra el código del fichero `listado.php` que es la vista encargada de mostrar todas las ofertas.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Listado de ofertas</title>
  </head>
  <body>
    <h1>Pizzeria Peachepe</h1>
    <a href=" ../Controller/nuevaOferta.php">Nueva oferta</a>
    <hr>
    <?php
      foreach($data['ofertas'] as $oferta) {
        ?>
        <h3><?=$oferta->getTitulo()?></h3>
        <br>
        <p><?=$oferta->getDescripcion()?></p><br>
        <a href=" ../Controller/borraOferta.php?id=<?=$oferta->getId()?>">Borrar</a>
      <?php
      }
    ?>
  </body>
</html>
```

El otro fichero de vista - `formularioOferta.php` - es el que muestra un formulario para introducir los datos de las nuevas ofertas. El código de este fichero se muestra a continuación.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <form action="../../Controller/grabaOferta.php" enctype="multipart/form-data" method="POST">
      <h3>Título</h3>
      <input type="text" size="40" name="titulo">
      <h3>Imagen</h3>
      <input type="file" id="imagen" name="imagen">
      <br><h3>Descripción</h3>
      <textarea name="descripcion" cols="60" rows="6">
      </textarea><br>
      <input type="submit" value="Aceptar">
    </form>
  </body>
</html>

```

10.3 El controlador

El controlador contiene lo que se da en llamar la lógica de negocio de la aplicación. Por ejemplo, en una aplicación de gestión de nóminas, el cálculo del sueldo bruto a partir de las horas trabajadas formaría parte de esa lógica de negocio. También es el controlador el encargado de juntar todas las piezas haciendo uso del modelo y dando la orden de cargar las vistas.

El fichero `index.php` del controlador obtiene todos los datos de las ofertas y se los envía a la vista `listado.php`.

```

<?php
require_once '../Model/Oferta.php';

// Obtiene todas las ofertas
$data['ofertas'] = Oferta::getOfertas();

// Carga la vista de listado
include '../View/listado.php';

```

El fichero `nuevaOferta.php` simplemente da la orden de cargar la vista que contiene el formulario donde se introducirán los datos de la nueva oferta.

```
<?php
// Carga la vista del formulario de alta de oferta
include '../View/formularioOferta.php';
```

El fichero grabaOferta.php recoge los datos enviados por el formulario y hace uso del modelo para grabar la información en la base de datos.

```
<?php
require_once '../Model/Oferta.php';

// sube la imagen al servidor
move_uploaded_file($_FILES["imagen"]["tmp_name"], "../View/images/" . $_FILES["imagen"][\
"name"]);

// inserta la oferta en la base de datos
$ofertaAux = new Oferta("", $_POST['titulo'], $_FILES["imagen"]["name"], $_POST['descrip\
cion']);
$ofertaAux->insert();
header("Location: index.php");
```

Por último, el fichero borraOferta.php elimina de la base de datos la oferta seleccionada.

```
<?php
require_once '../Model/Oferta.php';
$ofertaAux = new Oferta($_GET['id']);
$ofertaAux->delete();
header("Location: index.php");
```

10.4 Ejercicios



Ejercicio 1

Crea un blog siguiendo las pautas que se marcan a continuación:

- a) En un blog hay como mínimo una cabecera, una serie de artículos y un pie de página.
- b) Los artículos se almacenan en una base de datos. Sobre cada artículo se debe saber al menos el título, la fecha de publicación (o fecha y hora) y el contenido. Además cada artículo tendrá un identificador o código único (puede ser un código que se auto-incremente).
- c) El identificador puede ser un número que se vaya incrementando él solo.
- d) La fecha se puede coger del sistema cuando se graba un nuevo artículo.
- e) Crea la clase `BlogDB` para aislar los datos de la conexión a la base de datos donde se guardan los artículos.
- f) Crea la clase `Articulo` con los mismos atributos que campos hay en la tabla `articulo` de la base de datos. Esta clase debe tener implementado el constructor y opcionalmente los *getter* y *setter* (se pueden crear de forma automática con **Alt + Insert**).
- g) La clase `Articulo` tendrá también los métodos `insert` y `delete`, que deben insertar y borrar respectivamente un artículo de la base de datos.
- h) La clase `Articulo` debe tener también un método de clase `getArticulos()` que devuelva en un array todos los artículos de la base de datos.



Ejercicio 2

Mejora el blog de tal forma que se permita la modificación de un artículo.



Ejercicio 3

Añade estilos a la aplicación para hacerla más atractiva.

11. Twig - Motor de plantillas

11.1 Introducción

Twig es un motor de plantillas para programas escritos en PHP. En una aplicación que sigue el [Modelo Vista Controlador](#), Twig sirve para implementar la vista, es decir, la interfaz de usuario.

La sintaxis de Twig es muy clara y concisa por lo que para un diseñador, por regla general, es más fácil utilizar Twig que introducir código PHP en su página.

Twig es el motor de plantillas que se utiliza por defecto en herramientas de desarrollo muy potentes como **Drupal 8** o **Symfony**. También es posible utilizarlo en otros frameworks como **Laravel**, **Codeigniter** o **Yii** y, en general, en casi cualquier framework PHP.

Este motor de plantillas está desarrollado por [SensioLabs](#)¹, que es la misma empresa que desarrolla el framework [Symfony](#)².

Twig se puede descargar desde su [página oficial](#)³. Este sitio web es también un buen lugar donde obtener información y resolver dudas sobre este motor de plantillas.

11.2 “Hola mundo” con Twig

Crearemos un sencillo “Hola mundo” que nos servirá para conocer cómo se incluye Twig en un proyecto y cómo se usa.

En primer lugar, arranca Netbeans, crea un nuevo proyecto (Ctrl + Mayús + N) y dale el nombre `SaludoTwig` por ejemplo.

Descarga la [última versión de Twig desde GitHub](#)⁴. Descomprime el fichero zip y renombra la carpeta a `twig`. Copia la carpeta al directorio raíz de tu proyecto.

La estructura de ficheros y directorios quedará como se indica a continuación:

```
.
├─ index.php
├─ templates
│   └─ hola.html.twig
└─ twig
    (...)
```

¹<https://sensiolabs.com/>

²<http://symfony.com/>

³<http://twig.sensiolabs.org/>

⁴<https://github.com/twigphp/Twig>

No se muestra el contenido de la carpeta twig porque resulta irrelevante en este ejemplo.

Todas las plantillas Twig tienen la extensión `html.twig` y se guardan en la carpeta `templates`.

El fichero `index.php` tiene como único objetivo cargar la plantilla `hola.html.twig`. A continuación se muestra el código fuente.

```
<?php
require_once 'twig/lib/Twig/Autoloader.php';
Twig_Autoloader::register();
$loader = new Twig_Loader_Filesystem(__DIR__.'/templates');
$twig = new Twig_Environment($loader);
echo $twig->render('hola.html.twig', []);
```

Fíjate en la siguiente línea:

```
echo $twig->render('hola.html.twig', []);
```

A la función `render()` hay que pasarle como parámetro el nombre de la plantilla que se quiere cargar. El array vacío que se pasa también como parámetro nos está indicando que no se pasa ningún dato a la plantilla.

El fichero `hola.html.twig` contiene únicamente código HTML. Se muestra a continuación.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <h1> ¡Hola mundo! </h1>
  </body>
</html>
```

11.3 Paso de información a las plantillas

Como vimos en el [Capítulo 10: Modelo Vista Controlador](#), la vista consta de código HTML y normalmente contiene una serie de “huecos” donde va la información que le envía el controlador. En Twig, esos huecos se indican con los delimitadores `{{ y }}`.

Veamos el ejemplo incluido en la carpeta `SaludaTwig`. Desde el fichero `index.php`

```
<?php
require_once 'twig/lib/Twig/Autoloader.php';
Twig_Autoloader::register();
$loader = new Twig_Loader_Filesystem(__DIR__.'/templates');
$twig = new Twig_Environment($loader);
echo $twig->render('saluda.html.twig', ['saludo' => 'hola', 'x' => 24]);
```

Observa la siguiente línea:

```
echo $twig->render('saluda.html.twig', ['saludo' => 'hola', 'x' => 24]);
```

Al llamar a la función `render`, se pasan dos parámetros, igual que en el ejemplo anterior. El primer parámetro es el nombre de la plantilla que se quiere cargar y el segundo parámetro es un array asociativo con todos los datos. En este caso, pasamos `saludo` que vale `hola` y `x` que vale `24`.

El código de la plantilla `saluda.html.twig` se muestra a continuación.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    Probando Twig...<br>
    {{ saludo }}, la variable x vale {{ x }}, por si no lo sabías.
  </body>
</html>
```

Normalmente la información que hay que pasar a la vista es algo más que dos valores. El array que hay que incluir como parámetro en la función `render` se suele preparar de antemano con todos los datos necesarios.

Veamos el ejemplo contenido en la carpeta `DatosDeUsuarioTwig01`. En un array llamado `$datos` se prepara toda la información de un determinado usuario: DNI, nombre, etc. Posteriormente esta información será recuperada en la vista.

A continuación se muestra el código del fichero `index.php`.

```
<?php
require_once 'twig/lib/Twig/Autoloader.php';
Twig_Autoloader::register();
$loader = new Twig_Loader_Filesystem(__DIR__.'/templates');
$twig = new Twig_Environment($loader);

$datos = [
    'nombre' => 'Elena',
    'apellido1' => 'Nito',
    'apellido2' => 'Del Bosque',
    'dni' => '1234567X',
];

echo $twig->render('infoUsuario.html.twig', $datos);
```

El fichero de la vista, es decir `infoUsuario.html.twig`, se muestra a continuación.

```
{# Plantilla para mostrar información del usuario #}
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <h2>Información de usuario</h2>
    <b>Nombre:</b> {{ nombre }}<br>
    <b>Primer apellido:</b> {{ apellido1 }}<br>
    <b>Segundo apellido:</b> {{ apellido2 }}<br>
    <b>DNI:</b> {{ dni }}<br>
  </body>
</html>
```

Observa que se pueden incluir comentarios en las plantillas Twig mediante los delimitadores `{# y #}`.

La información contenida en el array que se pasa a la plantilla puede ser aún más compleja. Fíjate en el siguiente array (el código está incluido en la carpeta `DatosDeUsuarioTwig02`).

```
$datos = [
    titulo => 'Información de usuario',
    usuario => array(
        'nombre' => 'Elena',
        'apellido1' => 'Nito',
        'apellido2' => 'Del Bosque',
        'dni' => '1234567X')
];
```

Ahora el array `$datos` contiene un título que se mostrará en la página así como la información de un usuario. A su vez, la información del usuario está dividida en varios componentes. A continuación se muestra la plantilla que permite extraer toda esa información.

```
{# Plantilla para mostrar información del usuario #}
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <h2>{{ titulo }}</h2>
        <b>Nombre:</b> {{ usuario.nombre }}<br>
        <b>Primer apellido:</b> {{ usuario.apellido1 }}<br>
        <b>Segundo apellido:</b> {{ usuario.apellido2 }}<br>
        <b>DNI:</b> {{ usuario.dni }}<br>
    </body>
</html>
```

Observa que se utiliza el punto (.) para acceder a un dato que a su vez está dentro de otro; por ejemplo para acceder al DNI del usuario se utiliza `{{ usuario.dni }}`.

Ahora imagina que en lugar de los datos sobre un único usuario tenemos los datos de varios usuarios. Incluso se puede dar la situación de que no se sepa a priori el número de usuarios totales.

A continuación se muestra el código correspondiente al `index.php` incluido en la carpeta `DatosDeUsuarioT-wig03`.

```
<?php
require_once 'twig/lib/Twig/Autoloader.php';
Twig_Autoloader::register();
$loader = new Twig_Loader_Filesystem(__DIR__.'/templates');
$twig = new Twig_Environment($loader);

$infoUsuario = $twig->loadTemplate('infoUsuario.html.twig');

$datos = [
    titulo => 'Información de usuarios',
    usuarios => [
        [ 'nombre' => 'Elena',
          'apellido1' => 'Nito',
          'apellido2' => 'Del Bosque',
          'dni' => '1234567X' ],
        [ 'nombre' => 'Leandro',
          'apellido1' => 'Gado',
          'dni' => '3546732Q' ],
    ]
];

echo $twig->render('infoUsuario.html.twig', $datos);
```

Observa que ahora, en el array `$datos` hay dos elementos, por un lado está el título y por otro lado está la información de los usuarios; esta vez hay dos usuarios y, al igual que en los ejemplos anteriores, cada usuario tiene una serie de datos.

En la plantilla utilizaremos un bucle `for` para recorrer todos los usuarios. Veamos la plantilla completa.

```
{# Plantilla para mostrar información del usuario #}
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>

        <h2>{{titulo}}</h2>

        {% for usuario in usuarios %}
            <b>Nombre:</b> {{ usuario.nombre }}<br>
            <b>Primer apellido:</b> {{ usuario.apellido1 }}<br>
            {% if usuario.apellido2 is defined %}
                <b>Segundo apellido:</b> {{ usuario.apellido2 }}<br>
            {% endif %}
        {% endfor %}
```

```

        <b>DNI:</b> {{ usuario.dni }}<br>
    <hr>
{% endfor %}
</body>
</html>

```

El bucle comienza con la siguiente línea:

```
{% for usuario in usuarios %}
```

Se trata de un bucle del tipo `foreach` en el que se recorre el dato `usuarios` elemento a elemento desde el principio hasta el fin y cada uno de esos elementos contiene, a su vez, el DNI, los apellidos, etc.

Observa que para extraer el nombre de un usuario, escribimos la siguiente línea (exactamente igual que en el ejemplo anterior):

```
<b>Nombre:</b> {{ usuario.nombre }}<br>
```

11.4 Uso de Twig en el Modelo Vista Controlador

El motor de plantillas Twig encaja como un guante en el patrón Modelo Vista Controlador. La carpeta correspondiente a la librería Twig se suele colocar dentro de la carpeta `Controller` junto con otras librerías que se utilicen en el proyecto. Como habrás podido imaginar, las plantillas se guardan en la carpeta `View`.

El ejemplo `SumaTwig01` es una aplicación que simplemente suma dos números. La estructura de ficheros y directorios es la siguiente:

```

.
├── Controller
│   ├── index.php
│   ├── twig
│   └── (...)
├── index.php
├── Model
└── View
    ├── formulario.html.twig
    └── resultado.html.twig

```

El controlador contiene la lógica de negocio de la aplicación, por tanto será el encargado de sumar los números, así como de cargar las vistas necesarias en cada momento. Primero deberá cargar la vista que recoge los datos (un formulario) y luego tendrá que cargar la vista que muestra el resultado de la suma.

A continuación se muestra el código del fichero `index.php` del controlador.

```
<?php
require_once 'twig/lib/Twig/Autoloader.php';
Twig_Autoloader::register();
$loader = new Twig_Loader_Filesystem(__DIR__.'../View');
$twig = new Twig_Environment($loader);

if (!isset($_GET['x'])) {
    echo $twig->render('formulario.html.twig', []);
} else {
    $suma = $_GET['x'] + $_GET['y'];
    echo $twig->render('resultado.html.twig', ['suma' => $suma]);
}
```

En la primera carga del fichero index.php del controlador no se tienen los números que hay que sumar, por tanto hay que cargar la vista correspondiente al formulario que recoge los datos. Esta vista es formulario.html.twig y se muestra a continuación.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <form method="get">
            <input type="number" name="x" required="">+
            <input type="number" name="y" required="">
            <input type="submit" value="Sumar">
        </form>
    </body>
</html>
```

Una vez recogidos los datos desde el controlador, éste realiza la suma y se la envía a otra vista para que muestre el resultado. Esta vista es resultado.html.twig y se muestra a continuación.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>

        {{ suma }}
```

```

</body>
</html>

```

Esta aplicación se puede simplificar aún más si cabe. Se pueden leer los datos y mostrar el resultado en una única vista. El código de esta segunda versión de la aplicación que suma dos números está en la carpeta SumaTwig02. El fichero `index.php` del controlador se quedaría como se muestra a continuación.

```

<?php
require_once 'twig/lib/Twig/Autoloader.php';
Twig_Autoloader::register();
$loader = new Twig_Loader_Filesystem(__DIR__.'../View');
$twig = new Twig_Environment($loader);

$x = $_GET['x'];
$y = $_GET['y'];
echo $twig->render('formulario.html.twig', ['x' => $x, 'y' => $y, 'suma' => $x + $y]);

```

Ahora tenemos un sola vista, el fichero `formulario.html.twig`, que tiene la doble función de recoger los datos y de mostrar el resultado de la suma.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <form method="get">
      <input type="number" name="x" value="{{ x }}"> +
      <input type="number" name="y" value="{{ y }}"> = {{ suma }}<br>
      <input type="submit" value="Sumar">
    </form>
  </body>
</html>

```

11.5 Herencia de plantillas

La herencia es una de las características más importantes de Twig y facilita dos aspectos fundamentales en el desarrollo de un sitio web. Por un lado ahorra repetir código HTML de tal forma que todas las líneas que tienen una determinada función está solo en un sitio (no hay que copiar y pegar). Por otro lado, dota de una estructura coherente a las plantillas correspondientes a la interfaz del sitio web.

Ilustraremos la herencia con Twig con el ejemplo contenido en la carpeta `HerenciaTwig`. Vamos a centrarnos en los ficheros de plantillas - están dentro de la carpeta `View` - que son los que se listan a continuación.


```
asignaturas.html.twig
base.html.twig
contacto.html.twig
principal.html.twig
sobre.html.twig
```

La **plantilla padre** de todas las demás es `base.html.twig`. Cuando desde el controlador se manda visualizar una **plantilla hija**, se visualiza automáticamente la **plantilla padre** de ésta.

Veamos la estructura del archivo `base.html.twig`.

```
<!DOCTYPE html PUBLIC ...

(...)

<body>

(...)

{% block contenidoIzquierda %}{% endblock %}

(...)

{% block menu %}
<!-- menú por defecto -->
<ul>
  <li><a href="index.php">Principal</a></li>
  <li><a href="asignaturas.php">Asignaturas</a></li>
  <li><a href="sobre.php">Sobre nosotros</a></li>
  <li><a href="contacto.php">Contacto</a></li>
</ul>
{% endblock %}

<h2>Patrocinadores</h2>
<ul class="sponsors">

  {% block patrocinadores %}{% endblock %}

</ul>

(...)

</body>
</html>
```

Observa que hay varios bloques delimitados por las etiquetas `{% block nombreDeBloque %}` y `{% endblock %}`

%}. Todo el código HTML común está en este fichero `base.html.twig` y los bloques indican que se dejan huecos para que sean las **plantillas hijas** las que los rellenen: `asignaturas.html.twig`, `contacto.html.twig`, `principal.html.twig` y `sobre.html.twig`.

Por ejemplo, la siguiente línea define el bloque `contenidoIzquierda` que es el contenido que aparecerá en el lado izquierdo de la página.

```
{% block contenidoIzquierda %}{% endblock %}
```

Por tanto, en las **plantillas hijas** debe haber un bloque con este nombre que rellene este hueco.

Ahora observa este otro bloque dentro de `base.html.twig`:

```
{% block menu %}
<!-- menú por defecto -->
<ul>
    <li><a href="index.php">Principal</a></li>
    <li><a href="asignaturas.php">Asignaturas</a></li>
    <li><a href="sobre.php">Sobre nosotros</a></li>
    <li><a href="contacto.php">Contacto</a></li>
</ul>
{% endblock %}
```

Fíjate que esta vez, el bloque contiene código HTML. Esto quiere decir que si la **plantilla hija** define el bloque `menu`, el código que contenga rellenará - en este caso sustituirá - el bloque en el fichero `base.html.twig`. Pero si la **plantilla hija** no contiene el bloque `menu`, se quedará el código por defecto mostrado arriba.

Veamos ahora alguna de las **plantillas hija**, por ejemplo `contacto.html.twig`.

```
{% extends 'base.html.twig' %}

{% block contenidoIzquierda %}
<h2>Contact information<br />
    <span> March 16, 2010 | Posted by Owner | Filed under templates, internet</span> </h2>
    <p>Pellentesque elementum, felis nec vulputate facilisis, ligula mauris suscipit mi, id \
imperdiet metus quam sed erat. Mauris lectus ligula, scelerisque quis cursus eu, lacinia e\
u magna. Donec et orci justo. Donec id quam turpis. Curabitur sit amet fermentum turpis. E\
tiam blandit accumsan purus ut sodales. Maecenas ante risus, facilisis vel semper a, venen\
atis ut dui. </p>
    <h2>Contact Form</h2>
    <form action="#" method="post" id="contactform">
        <ol>
            <li>
                <label for="name">Your Name*</label>
                <input id="name" name="name" class="text" />
            </li>
        </ol>
    </form>
{% endblock %}
```

```

    <li>
        <label for="email">E-Mail*</label>
        <input id="email" name="email" class="text" />
    </li>
    <li>
        <label for="company">Website</label>
        <input id="company" name="company" class="text" />
    </li>
    <li>
        <label for="message">Your Message*</label>
        <textarea id="message" name="message" rows="6" cols="50"></textarea>
    </li>
    <li class="buttons">
        <input type="submit" name="imageField" id="imageField" value="Send message" class\
="send" />
        <div class="clr"></div>
    </li>
</ol>
</form>
{% endblock %}

```

{# No defino el menú, por tanto se muestra el menú definido por defecto en base.twig #} \

```

{% block patrocinadores %}
    <li class="sponsors"><a href="http://www.dreamtemplate.com">DreamTemplate</a><br />
    Over 6,000+ Premium Web Templates</li>
    <li class="sponsors"><a href="http://www.templatesold.com/">TemplateSOLD</a><br />
    Premium WordPress & Joomla Themes</li>
    <li class="sponsors"><a href="http://www.imhosted.com">ImHosted.com</a><br />
    Affordable Web Hosting Provider</li>
    <li class="sponsors"><a href="http://www.csshub.com/">CSS Hub</a><br />
    Premium CSS Templates</li>
    <li class="sponsors"><a href="http://www.evrsoft.com">Evrsoft</a><br />
    Website Builder Software & Tools</li>
    <li class="sponsors"><a href="http://www.myvectorstore.com">MyVectorStore</a><br />
    Royalty Free Stock Icons</li>
{% endblock %}

```

La siguiente línea es la que indica que la plantilla actual, es decir `contacto.html.twig` es una **plantilla hija** de `base.html.twig`.

```
{% extends 'base.html.twig' %}
```

Observa que se han definido los bloques `contenidoIzquierda` y `patrocinadores` que eran huecos que había

dejado la **plantilla padre** sin rellenar. No se ha definido el bloque `menu`, por tanto se visualizará el que había por defecto en `base.html.twig`.

11.6 Ejercicios



Ejercicio 1

Actualiza el blog realizado anteriormente de tal forma que todas las vistas estén implementadas en Twig.



Ejercicio 2

Mejora el ejercicio anterior incorporando la herencia en las plantillas para no repetir código de la cabecera, el pie de página, etc.