# Tema 2: Programación basada en lenguajes de marcas con código embebido

## ¿Qué aprenderás?

- La sintaxis de las construcciones condicionales en PHP.
- La sintaxis de las construcciones iterativas en PHP.
- Cómo trabaja PHP con los arrays.
- Cómo crear y usar funciones en PHP.
- El uso de formularios para recoger los datos del usuario y tratarlos en PHP.

## ¿Sabías que…?

- La construcción switch es equivalente a los bloques if-elseif, pero se usa por su sintaxis más clara.
- Los bucles while siempre pueden convertirse en bucles for. En cambio, al revés, no siempre es posible.
- En los bucles do-while como mínimo realizamos una iteración. En los bucles while puede que no se ejecute ni una iteración.

# 1. Construcciones condicionales

## 1.1. Introducción

Las construcciones condicionales son aquellas que ejecutan un bloque de instrucciones únicamente si se cumplen ciertas condiciones. Las dos construcciones condicionales que usaremos en PHP son:

- Bloque if: establece una condición y la prueba. Si la condición es verdadera, se ejecuta el bloque de instrucciones.
- Bloque switch: establece una lista de condiciones alternativas. Prueba la condición para ver si es verdadera y ejecuta el bloque de instrucciones apropiado.

## 1.2. Bloque if

La instrucción if pregunta si ciertas condiciones se cumplen. Un bloque de instrucciones se ejecutará dependiendo de cuáles son las condiciones que se cumplen. Las condiciones que se evalúan suelen ser comparaciones, utilizando los operadores de comparación vistos en el tema 1. Las condiciones que se evaluarán en un if pueden combinar varias comparaciones con los operadores AND, OR y XOR.

El formato de un bloque condicional if es el siguiente:

```
if (condición) {
     bloque de instrucciones;
}
elseif (condición) {
     bloque de instrucciones;
}
else {
     bloque de instrucciones;
}
```

Como vemos, el bloque if consta de tres secciones:

- if: esta sección es obligatoria. Comprueba la condición.
    - Si la condición es cierta se ejecuta el bloque de enunciados. Una vez se hayan ejecutado, el programa continuará ejecutando el programa pasando por alto las secciones elseif o else que pudieran suceder a la sección if.
    - Si la condición es falsa el bloque de enunciados no se ejecuta. El programa pasa a la siguiente instrucción, la cual puede ser una sección elseif o else.
- elseif: esta sección es opcional. Comprueba una condición. Podemos usar más de una sección elseif si lo deseamos.

- Si la condición es cierta se ejecuta el bloque de enunciados. Una vez se hayan ejecutado, el programa continuará ejecutando el programa pasando por alto las secciones elseif o else que pudieran suceder a la sección elseif.
- Si la condición es falsa el bloque de enunciados no se ejecuta. El programa pasa a la siguiente instrucción, la cual puede ser una sección elseif o else.

- else: esta sección es opcional. Sólo podemos tener una sección else por cada sección if. Esta sección no prueba una condición. Su bloque de instrucciones se ejecutará si la condición del if y las condiciones de todas las secciones elseif son falsas.

En el siguiente ejemplo, queremos que nuestra página muestre la nota de un alumno según el siguiente criterio: 10 – Matrícula de honor, 9 – Excelente, 8 y 7 – Notable, 6 – Bien, 5 – Suficiente, 4 y 3 – Insuficiente, notas por debajo Muy deficiente.

```
if ($nota == 10) {
     echo "Matrícula de honor";
}
elseif ($nota == 9) {
     echo "Excelente";
}
elseif ($nota == 8 || $nota == 7) {
     echo "Notable";
}
elseif ($nota == 6) {
     echo "Bien";
}
elseif ($nota == 5) {
     echo "Suficiente";
}
elseif ($nota == 4 || $nota == 3) {
     echo "Insuficiente";
}
elseif ($nota >= 0 && $nota <= 10) {
     echo "Muy deficiente";
}
else {
     echo "Nota incorrecta";
}
```

Véanse las condiciones que se han utilizado en los bloques if y elseif. Se han utilizado los operadores de igualdad (==), y en algunos casos se han combinado con el operador OR (||) o con el operador AND (&&). Recordemos que, con el operador OR, para que la condición sea cierta, basta con que una única comparación unida por el operador OR sea cierta. En cambio, con el operador AND todas las comparaciones deben ser ciertas.

En el último bloque elseif la condición para ejecutar el bloque asociado es que la nota sea mayor o igual a 0 y menor o igual que 10. Se podría pensar que con una nota de 10, por ejemplo, se ejecutaría este bloque. Esto no es así ya que antes de verificarse la condición de este bloque elseif, el programa ya ha verificado las comparaciones de los bloques anteriores.

Por tanto, si se hubiera cumplido alguna condición anterior, ya no se miraría si la nota es mayor o igual que 0 o menor o igual a 10.

El bloque else se ha introducido para que se ejecute cuando el valor de la variable $nota no esté en el rango de 0 a 10. En el bloque elseif anterior al bloque else se ha añadido en la condición la comparación $nota <= 10. De no ponerla, con una nota igual a 12, nos saldría el mensaje "Muy deficiente". Esto no lo queremos, debe salir el mensaje "Nota incorrecta". De ahí que la condición del último bloque elseif tenga esa forma.

Cuando el bloque a ejecutarse por cualquier sección del enunciado condicional if contiene una sola instrucción, las llaves no son necesarias. El ejemplo anterior podríamos haberlo escrito así:

```
if ($nota == 10)
     echo "Matrícula de honor";
elseif ($nota == 9)
     echo "Excelente";
elseif ($nota == 8 || $nota == 7)
     echo "Notable";
elseif ($nota == 6)
     echo "Bien";
elseif ($nota == 5)
     echo "Suficiente";
elseif ($nota == 4 || $nota == 3)
     echo "Insuficiente";
elseif ($nota >= 0 && $nota <= 10)
     echo "Muy deficiente";
else
     echo "Nota incorrecta";
```

De esta forma podemos ahorrarnos algunos caracteres, pero el código puede ser un poco más confuso. De todas formas, hay que tener claro que los bloques de instrucciones de un if, cuando hay más de una instrucción, deben ir entre llaves SIEMPRE.

Podemos meter bloques if dentro de otros bloques if. A esto se le llama anidar. Veamos el siguiente ejemplo:

```
if ($paisCliente == "España") {
     if ($direccionEmail != "") {
          $metodoContacto = "email";
     }
     else {
          $metodoContacto = "carta";
     }
}
else {
     echo "No se necesita";
}
```

El código hace lo siguiente: mira si el cliente es de España. Si es así, mira si la dirección de email no está vacía (es decir, es diferente de una cadena vacía). Si es así, el método de contacto será "email". Sino, el método de contacto será "carta". Si el país del cliente no es España, en el método de contacto pondremos "No se necesita".

## 1.3. Bloque switch

Un bloque switch es equivalente a una serie de sentencias if. Su uso típico es cuando queremos ejecutar diferente código en función del valor de una variable, que puede tener varios valores diferentes.

El bloque switch parte del valor que almacena una variable. En función de ese valor ejecutará unas instrucciones u otras, contenidas en bloques case, hasta llegar al enunciado break o al final del bloque switch. Si no hay una sección case para el valor de la variable estudiada, se ejecutarán las instrucciones del bloque default. Veamos el código:

```
switch ($nombreVariable) {
     case valor1:
          bloque de instrucciones;
          break;
     case valor2:
          bloque de instrucciones;
          break;
     ...
     default:
          bloque de instrucciones;
}
```

Obsérvese que cada sección case se termina con la instrucción break. Ésta se debe poner para que el programa para la ejecución del switch. Si no se pone, al terminar de ejecutar las instrucciones de un bloque case, a continuación se ejecutarían las instrucciones del siguiente bloque case.

La sección default es opcional. Si se pone, se acostumbra a colocarse al final, aunque se puede poner en cualquier parte. Al ponerla al final, no es necesario incluir la instrucción break, ya que las instrucciones que se ejecutarían después serían las de fuera del bloque switch, cosa totalmente normal.

En el siguiente ejemplo se establece una matrícula en función de la provincia:

```
switch ($provincia) {
     case "Barcelona":
          $matricula = "B";
          break;
     case "Madrid":
          $matricula = "M";
          break;
     case "Valencia":
          $matricula = "V";
          break;
     case "Sevilla":
          $matricula = "S";
          break;
     default:
          $matricula = "Otra";
}
```

En el ejemplo anterior, si el valor de $provincia es Barcelona, en la variable $matricula se pone una B. Si la variable $provincia contiene Madrid, la variable $matricula valdrá M... y se hace lo mismo con las valores Valencia y Sevilla para la variable $provincia. En caso de tener un valor distinto a los anteriores, se ejecutará el bloque default, asignando a la variable $matricula el valor Otra.

Veamos otro ejemplo:

```
switch ($nota) {
     case 0:
     case 1:
     case 2:
          $evaluacion = "Muy deficiente";
          break;
     case 3:
     case 4:
          $evaluacion = "Insuficiente";
          break;
     case 5:
          $evaluacion = "Suficiente";
          break;
     case 6:
          $evaluacion = "Bien";
          break;
     case 7:
     case 8:
          $evaluacion = "Notable";
          break;
     case 9:
          $evaluacion = "Excelente";
          break;
     case 10:
          $evaluacion = "Matrícula de honor";
          break;
     default:
          echo "Nota incorrecta";
}
```

Se ha reescrito el ejemplo que habíamos hecho con bloques if, elseif, pero esta vez con un switch. Se puede ver su equivalencia.

Llama la atención que hay secciones case "vacías". Esto es porque, por ejemplo, para los valores 0, 1 y 2 de la variable $nota queremos ejecutar lo mismo. Por tanto, ponemos los case asociados a los valores 0 y 1 vacíos, sin ningún break. El primer break lo vemos en el case asociado al valor 2. De esta forma, si la variable $nota es 0, se ejecutará las instrucciones hasta el primer break, es decir, $evaluacion = "Muy deficiente".

# 2. Construcciones iterativas

## 2.1. Introducción

Las construcciones iterativas, o bucles, establecen un bloque de instrucciones que se repiten. A veces, el bucle se repite un número específico de veces, y otras el bucle se repite hasta que cierta condición se cumple.

En PHP existen los siguientes tipos de bucles:

- Bucle for: establece un contador y repite un bloque de instrucciones hasta que el contador alcanza un número específico.
- Bucle while: fija una condición y la verifica. Si es cierta, se repite el bloque de instrucciones.
- Bucle do…while: establece una condición, ejecuta un bloque de instrucciones, y verifica la condición. Si es cierta, repite el bloque de instrucciones.

## 2.2. Bucle for

El bucle for básico se basa en un contador. Tenemos que fijar los valores inicial y final para el contador, además de cómo el contador se incrementa o decrementa. La sintaxis es la siguiente:

```
for (valor inicial;condición;incremento) {
     bloque de instrucciones;
}
```

Las partes que se escriben entre paréntesis son las siguientes:

- Valor inicial: se inicializa una variable que representará el contador a un valor inicial. Se hace una única vez, al iniciar el bucle.
- Condición: enunciado que establece cuándo se tiene que ejecutar el bucle. Siempre que este enunciado sea verdadero, el bloque de instrucciones sigue repitiéndose. Cuando este enunciado no es verdadero, el bucle termina. Esta condición se prueba en cada iteración del bucle.
- Incremento: instrucción que incrementa el contador. También pueden usarse operaciones de decremento. Se ejecutará justo después de que se haya ejecutado el bloque de instrucciones situados dentro del for.

Por ejemplo, queremos sumar los 10 primeros números:

```
$suma = 0;
for ($i=1;$i<=10;$i++) {
     $suma = $suma + $i;
}
```

Utilizamos la variable $i como contador, inicializándola a 1. Queremos que el bucle se repita mientras el valor de $i sea menor o igual que 10. Iremos incrementando de uno en uno el

valor de la variable $i para avanzar en nuestro bucle. Dentro del bucle, actualizaremos la variable $suma añadiéndole el valor de la variable $i. Cada vez que se ejecute el bucle, la variable $i tendrá un valor diferente (se incrementará el valor que tenía en uno).

Recordemos que la operación de incremento se ejecuta después de ejecutar todas las instrucciones de dentro del bucle. Además, la instrucción de valor inicial, se ejecuta una sola vez, al iniciar el bucle por primera vez.

Los bucles for se suelen utilizar para movernos por un array. En el fragmento de código siguiente, vamos a recorrer los valores del array $notas:

```
for ($i=0;$i<sizeof($notas);$i++) {
     echo $notas[$i].”<br>”;
}
```

La función sizeof(array) se usa para saber el número de elementos que contiene un array.

Hemos visto que el bucle for tiene tres secciones, que hemos llamado valor inicial, condición e incremento. Estas tres secciones tienen que separarse con un punto y coma (;). Cada sección puede contener tantos enunciados como sean necesarios, separados por comas. Cualquier sección puede estar vacía. Veamos el siguiente ejemplo:

```
for ($i=0, $j=1;$t<=4;$i++, $j++) {
     $t = $i + $j;
     echo $t.”<br>”;
}
```

El código anterior imprimirá los valores 1, 3 y 5.

## 2.3. Bucle while

El bucle while ejecuta un bloque de instrucciones siempre que se cumpla cierta condición. El funcionamiento del bucle while es el siguiente:

1. El programador establece una condición.
2. La condición se comprueba al inicio de cada iteración.
3. Si la condición es verdadera, se ejecuta el bloque de código dentro del bucle y se vuelve al inicio del while (o sea, a la comprobación de la condición). Si la condición no se cumple, el programa sigue con su ejecución en la instrucción posterior al bucle while.

La sintaxis de un bucle while es la siguiente:

```
while (condición) {
     bloque de instrucciones;
}
```

La condición será cualquier expresión que pueda ser cierta o falsa. Construiremos nuestras condiciones usando operadores de comparación combinándolos con los operadores AND, OR y XOR.

En el siguiente ejemplo vemos un bucle while que generará el output “Hola mundo!” cinco veces:

```
$veces = 0;
```

```
while ($veces != 5) {
     echo "Hola mundo!";
     $veces++;
}
```

El programa empieza con la inicialización de la variable $veces a 0. Entonces el programa comprobará la condición del bucle, si $veces es diferente a 5. Como 0 es diferente a 5, se ejecutarán las instrucciones que hay dentro del bucle. Es importante la instrucción $veces++. Cada vez que entramos en el bucle, hay que incrementar el valor de la variable $veces. De esta forma nos aseguramos que su valor llega a 5, por tanto llegará un momento en que la condición del bucle no se cumplirá, y el programa avanzará (en este ejemplo el programa terminará).

Un error común con los bucles while son los bucles infinitos, es decir, nuestro programa se queda ejecutando eternamente las instrucciones contenidas dentro del bucle (bueno, eternamente no, por defecto PHP está configurado para parar a los 30 segundos). Esto es debido a que la condición del while siempre se cumple. En el ejemplo anterior, ¿qué pasaría si quitáramos la instrucción $veces++? Estaríamos en un bucle infinito, pues la variable $veces siempre valdría 0, y 0 siempre es diferente de 5.

Una forma de asegurarnos de que no creamos un bucle infinito es incluir en el cuerpo del bucle una instrucción que modifique el valor de la variable que usamos para la condición. En nuestro caso, la variable $veces tiene que ir variando su valor hasta llegar a 5. Por tanto, basta con sumar 1 en cada iteración para llegar al final de la ejecución del bucle.

## 2.4. Bloque do…while

Los bucle do…while son muy parecidos a los bucles while. Establecemos una condición y en base a como se evalúe esa condición (si es cierta) se ejecutará un bloque de instrucciones. La diferencia con el bucle while es que con el do..while primero se ejecuta el bloque de instrucciones y, al terminar éstas, se comprueba la condición. Si es cierta, se vuelve a ejecutar el bloque de instrucciones, sino no.

Por tanto, podemos decir que los bucles do…while iteran como mínimo una vez, mientras que en los bucles while puede que el bloque de instrucciones no llegue a ejecutarse (la condición es falsa inicialmente).

La sintaxis de un bucle do…while es la siguiente:

```
do{
     bloque de instrucciones;
} while (condición);
```

Veamos cómo quedaría el ejemplo visto anteriomente, pero esta vez escrito con un bucle do…while:

```
$veces = 0;
do{
     echo "Hola mundo!";
     $veces++;
} while ($veces != 5);
```

## 2.5. Instrucciones para salir de un bucle

En ciertas ocasiones, querremos que nuestros programas salgan de la ejecución de un bucle. En PHP tenemos las siguientes instrucciones que podemos usar con ese fin:

- break: termina el bucle y el programa continúa con las instrucciones que el programa tenga después.
- continue: termina la ejecución de una iteración y vuelve a la comprobación del bucle, para ver si se ejecuta otra iteración o no.

Veamos unos ejemplos de uso de estas dos instrucciones:

```
$contador = 0;
while ($contador < 5){
     $contador++;
     if ($contador == 3){
          echo "break".<br>;
          break;
     }
     echo "Iteración = ".$contador."<br>";
}
echo "Final del bucle while";
```

El output que generará el código anterior será:

```
Iteración = 1
Iteración = 2
break
Final del bucle while
```

Vemos que al ejecutarse la instrucción break, el bucle termina su ejecución.

Ahora hagamos el mismo programa sustituyendo el break por un continue:

```
$contador = 0;
while ($contador < 5){
     $contador++;
     if ($contador == 3){
          echo "continue".<br>;
          continue;
     }
     echo "Iteración = ".$contador."<br>";
}
echo "Final del bucle while";
```

El output que generará el código anterior será:

```
Iteración = 1
Iteración = 2
continue
Iteración = 4
Iteración = 5
Final del bucle while
```

Vemos que pese a ejecutarse la instrucción continue, el bucle no termina su ejecución. Pero no se imprime la frase "Iteración = 3". Esto es debido a que la instrucción continue termina la ejecución de la iteración. Por tanto, las instrucciones que van después del continue, no se ejecutan. Después del continue el programa vuelve al inicio del bucle, a comprobar la condición.

# 3. Arrays

## 3.1. Introducción

Los arrays son variables complejas. Un array almacena un grupo de valores bajo un único nombre de variable. PHP nos provee de las herramientas necesarias para manejar, acceder y modificar la información almacenada en un array.

## 3.2. Creación de arrays

Hay varias formas de crear un array en PHP. La más simple es asignar un valor a una variable con corchetes ([]) al final de su nombre. Por ejemplo, vamos a guardar las tres notas de un alumno:

```
$notas[1] = 9;
$notas[2] = 8;
$notas[3] = 10;
```

Los enunciados anteriores crean un array que almacenan 3 valores. Es importante señalar que suponemos que no hemos hecho referencia anteriormente en nuestro programa a la variable $notas.

Un array se puede considerar como una lista de parejas de claves y valores. Para obtener un valor en particular, basta con especificar la clave entre corchetes. En el array anterior, las claves son números: 1, 2 y 3. Sin embargo, podemos usar también palabras como claves. Por ejemplo, los enunciados siguientes crean un arreglo de capitales de comunidades autonómicas:

```
$capitales['CAT'] = "Barcelona";
$capitales['AND'] = "Sevilla";
$capitales['ARA'] = "Zaragoza";
```

No es necesario que siempre que creemos un array asignemos claves a los valores. Podemos crear el array anterior de notas de la siguiente forma:

```
$notas[] = 9;
$notas[] = 8;
$notas[] = 10;
```

Cuando creamos el array de esta forma, a los valores se les asigna automáticamente claves que son números de serie, empezando con el número 0. Así pues, para mostrar el primer valor del array escribiríamos:

```
echo $notas[0];
```

Otra forma de crear un array es la siguiente:

```
$notas = array(9, 8, 10);
```

De esta forma creamos el mismo array que en el ejemplo anterior, asignando números como claves, empezando con 0. Podemos crear de forma parecida un array que tenga palabras como claves, por ejemplo:

```
$capitales = array("CAT" => "Barcelona", "AND" =>
"Sevilla", "ARA" => "Zaragoza");
```

## 3.3. Acceder a los valores de un array

Para visualizar los valores que almacena un array podemos utilizar la función echo, especificando entre corchetes la clave del valor que queremos mostrar:

```
echo $capitales['AND'];
```

Para ver la estructura y los valores de un array usamos la función print_r. Su uso es el siguiente:

```
print_r($capitales);
```

Este enunciado dará el siguiente output, mostrando la clave y los valores para cada elemento del array:

```
array ([CAT] => Barcelona [AND] => Sevilla [ARA] =>
Zaragoza)
```

También podemos guardar en una variable un valor del array:

```
$capitalCAT = $capitales['CAT'];
```

Si en la operación anterior intentamos acceder a una posición del array que no existe, obtendremos una advertencia. Por ejemplo:

```
$capitalCAT = $capitales['CA'];
Notice: Undefined index: CA in…
```

Hay una forma de obtener los valores de un array y almacenarlos automáticamente en diferentes variables. Se hace mediante la función list, que recibe como parámetros las variables en las que queremos guardar los valores del array. A continuación se muestra un fragmento de código, en que se declara un array con tres valores. Luego, mediante la función list, se guardan los dos primeros valores del array en dos variables, $var1 y $var2:

```
$miArray = array("valor1", "valor2", "valor3");
list($var1, $var2);
echo $var1, "<br>";
echo $var2, "<br>";
```

El primer echo que se hace, mostrará valor1 (seguido de un salto de línea), y el segundo echo mostrará valor2. Si quisiéramos obtener el también el tercer valor del array, basta con incluir una tercera variable como parámetro de la función list.

Por tanto, lo que hemos hecho en el fragmento de código anterior, es equivalente a:

```
$var1 = $miArray[0];
$var2 = $miArray[1];
```

Otra manera de obtener los valores de un array en diferentes variables es usando la función extract. Cada valor se copia en una variable con un nombre que se corresponde con la clave. Por ejemplo, si usamos el array de capitales, podemos hacer:

```
extract($capitales);
echo "Las capitales son ".$CAT." ".$AND." ".$ARA;
```

Nótese que se crean tres variables con los mismos nombres que las claves de los tres valores que guarda el array. Además, en el echo hemos utilizado el operador punto para concatenar diversas cadenas de caracteres.

## 3.4. Quitar valores a un array

Supongamos que tenemos el siguiente array que guarda nombres de personas:

```
$personas = array ("José", "Esteban", "María", "Lucía",
"Pedro");
```

El array $personas tiene 5 valores. Si decidimos que queremos eliminar el último valor, parece obvio hacer lo siguiente:

```
$personas[4] = "";
```

El enunciado anterior es incorrecto. Si bien estamos estableciendo una cadena vacía como valor de la quinta posición del array, seguiremos teniendo un array con cinco valores. Para elimina completamente un elemento del array, necesitamos destruirlo de la siguiente forma:

```
unset ($personas[4]);
```

Ahora nuestro array sólo cuenta con cuatro valores.

## 3.5. Recorrer un array

Como hemos visto los arrays permiten guardar varios valores bajo un mismo nombre de variable. La mayoría de veces vamos a querer hacer operaciones sobre cada uno de estos valores, como por ejemplo, guardarlos en una base de datos. Así pues, la operación más común con un array es la de iterar, es decir recorrer todos los valores del array y hacer algo con ellos.

Para iterar con cada valor de un array se suele usar la instrucción foreach: nos permite movernos automáticamente por el array, desde el inicio hasta el final, de valor en valor, ejecutando para cada uno un bloque de instrucciones.

La sintaxis de la instrucción foreach es la siguiente:

```
foreach ($nombreArray as $nombrevalor) {
     bloque de instrucciones;
}
```

Lo que hace este fragmento de código es recorrer el array $nombreArray, y en cada iteración guarda el valor del elemento actual en la variable $nombrevalor. Con este valor ejecuta el bloque de instrucciones, y cuando termina pasa al siguiente valor del array, hasta que no hay más.

Por ejemplo, vamos a hacer la media de los elementos de un array:

```
$miArray = array (1, 2, 3, 4);
$suma = 0;
foreach ($miArray as $valor){
     $suma = $suma + $valor;
}
$media = $suma / 4;
```

Con la instrucción foreach también podemos obtener la clave de cada posición del array, de la siguiente forma:

```
foreach ($nombreArray as $nombreclave => $nombrevalor) {
     bloque de instrucciones;
}
```

Si queremos modificar directamente los elementos del array dentro del bucle foreach, tenemos que poner el carácter & antes de $nombrevalor.

Por ejemplo, vamos a multiplicar por 2 todos los valores del array:

```
$miArray = array (1, 2, 3, 4);
foreach ($miArray as &$valor){
     $valor = $valor * 2;
}
```

Ahora los valores del vector anterior serán 2, 4, 6, 8. Sin poner el carácter &, los valores seguirían siendo los mismos que antes de ejecutar el bloque foreach.

PHP ofrece otra forma de movernos por un array, sin tener que utilizar la instrucción foreach. Esto se consigue con el uso de punteros, que señalaran a un elemento del array, y podremos mover a nuestro antojo. Para realizar los movimientos tenemos las siguientes funciones:

| Función | Descripción |
|---|---|
| current ($nombreArray) | Se refiere al valor que está señalando el puntero. Devuelve FALSE si no hay valor. |
| next ($nombreArray) | Mueve el puntero hasta el valor que está después del valor actual. Devuelve FALSE si no hay valor. |

| | |
|---|---|
| previous ($nombreArray) | Mueve el puntero hasta el valor que está antes de la ubicación actual del puntero. Devuelve FALSE si no hay valor. |
| end ($nombreArray) | Mueve el puntero hasta el último valor en el array. Devuelve FALSE si el array está vacío. |
| reset ($nombreArray) | Mueve el puntero hasta el primer valor en el array. Devuelve FALSE si el array está vacío. |

La primera vez que usemos la función current con un array, nos devolverá el primer valor.

Por ejemplo, vamos a mostrar las tres capitales de nuestro array $capitales:

```
$valor = current($capitales);
echo $valor."<br>";
$valor = next($capitales);
echo $valor."<br>";
$valor = next($capitales);
echo $valor."<br>";
```

Podemos ver que usando estas funciones ganamos flexibilidad en cuanto a los movimientos que podemos hacer sobre los valores de un array. Pero si vamos a acabar recorriéndolos todos, lo mejor es utilizar un bloque foreach.

## 3.6. Consultar el tamaño de un array

En ocasiones será útil saber el número de elementos que tiene un array, esto es, su tamaño. Para ello, los arrays en PHP disponen de la función `count()`. Vemos un ejemplo de su uso:

```
$miArray[0] = 10;
$miArray[1] = 20;
$miArray[2] = 30;
echo "El array tiene ".count($miArray)." posiciones<br>";
```

La función `count()` tiene un alias, es decir, otro forma de ejecutarla pero que en realidad hace lo mismo. Este alias es `sizeof`, y se usa de la siguiente forma:

```
$miArray = [10, 20, 30];
for($i=0;$i<sizeof($miArray);$i++){
     echo "$miArray[$i]<br/>";
}
```

## 3.7. Funciones útiles para arrays

El lenguaje PHP ofrece un amplio número de funciones ya predefinidas para trabajar con arrays. Algunas de ellas se pueden observar en la siguiente tabla:

| Función | Descripción |
|---|---|
| $array1 == $array2 | Devuelve cierto si los dos arrays tienen los mismos valores |
| array_merge($array1, $array2, …) | Devuelve un array resultado de combinar todos los valores de los arrays que la función recibe como parámetros |
| array_push($array, “valor1”, “valor2”, …) | Añade al final del array los elementos dados |
| array_pop($array) | Elimina el último elemento del array |
| in_array(“valor”, $array) | Comprueba si un array tiene un valor, devolviendo cierto o falso |
| implode(“,”, $array) | Une en un string los elementos de un array separados por comas |

## 3.8. Ordenar un array

PHP nos ofrece diversas funciones para ordenar los valores de un array. Las más comunes son sort y asort.

La función sort se utiliza para ordenar los valores de un array que tiene números como claves. La sintaxis es la siguiente:

```
$notas = array(9, 8, 10);
sort ($notas);
echo $notas[0];
echo $notas[1];
echo $notas[2];
```

Este código producirá el output 8, 9 y 10, mostrando los valores del array ordenados. La función sort funcionaría igualmente con un array que guardara cadenas de caracteres. Recordemos que las cadenas se ordenan teniendo en cuenta el código ASCII de cada carácter.

Si usamos la función sort para ordenar un array con palabras como claves, las claves cambiarán a números, y las palabras claves se perderán.

Para ordenar arrays que tengan palabras como claves, usaremos la función asort como sigue:

```
asort ($capitales);
```

Este enunciado clasifica las capitales por valores, y retiene la clave original para cada valor en lugar de asignar una clave numérica. Por ejemplo, si suponemos que inicialmente el array $capitales tiene esta información:

```
$capitales['ARA'] = "Zaragoza";
$capitales['CAT'] = "Barcelona";
$capitales['AND'] = "Sevilla";
```

Después de ejecutar la función asort($capitales), el array quedará así:

```
$capitales['CAT'] = "Barcelona";
$capitales['AND'] = "Sevilla";
$capitales['ARA'] = "Zaragoza";
```

Hay otras funciones de ordenación:

| Función | Descripción |
|---|---|
| rsort ($array) | Ordena según valor en orden inverso. Asigna números nuevos como claves. |
| arsort ($array) | Ordena según valor en orden inverso. Mantiene la misma clave. |
| ksort ($array) | Ordena según clave. |
| krsort ($array) | Ordena según clave en orden inverso. |
| usort ($array, función) | Ordena según función. |

## 3.9. Arrays multidimensionales

Los arrays multidimensionales pueden entenderse como un array que en cada posición guarda un array. Podríamos pensar en una matriz para situar los valores del array multidimensional. Cada valor se identifica por dos claves, filas y columna.

Veamos un ejemplo, en que queremos guardar las notas de una clase. Cada nota corresponde a un alumno concreto y a una asignatura concreta. Por tanto vamos a utilizar dos claves, el nombre del alumno y el nombre de la asignatura, para referenciar una nota:

```
$notas ['Carlos']['Mates'] = 8;
$notas ['Carlos']['Lengua'] = 10;
$notas ['Inés']['Mates'] = 6;
$notas ['Inés']['Lengua'] = 5;
$notas ['José']['Mates'] = 7;
$notas ['José']['Lengua'] = 4;
```

En el ejemplo anterior podemos ver la estructura de $notas como un array de arrays. $notas tiene claves Carlos, Inés y José. El valor para cada clave es un array con dos claves, Mates y Lengua.

$notas es un array bidimensional. El título de este apartado sugiere que en PHP se pueden construir arrays de tres, cuatro, cinco o más dimensiones.

El array bidimensional anterior también podemos declararlo como sigue:

```
$notas = array (
     'Carlos' => array ('Mates' => 8, 'Lengua' => 10),
     'Inés' => array ('Mates' => 6, 'Lengua' => 5),
     'José' => array ('Mates' => 7, 'Lengua' => 4)
);
```

Podemos obtener el valor de un array bidimensional de la siguiente forma:

```
$nota = $notas['Inés']['Lengua'];
```

Por supuesto, podremos recorrer el array bidimensional usando construcciones foreach. Necesitaremos un bloque foreach para cada array. Un enunciado forecha está dentro de otro enunciado foreach. Dicho de una forma más correcta, utilizaremos dos foreach anidados.

En el siguiente ejemplo, vamos a construir una tabla html que contenga las notas de cada alumno:

```
echo "<table border=1>";
foreach ($notas as $nombreAlumno => $notasAlumno){
     echo "<tr><td>".$nombreAlumno."</td>";
     foreach ($notasAlumno as $nombreModulo => $nota){
          echo "<td>".$nombreModulo.": ".$nota."</td>";
     }
     echo "</tr>";
}
echo "</table>";
```

Antes del primer foreach construimos la tabla con la etiqueta table de HTML. Como es un output que queremos que interprete el navegador, usamos el enunciado echo.

El primer foreach nos sirve para recorrer los alumnos de nuestro array bidimensional. Guardamos el nombre de cada alumno en la variable $nombreAlumno y su array de notas en la variable $notasAlumno. Para cada alumno creamos una nueva fila de la tabla, etiqueta tr, y una columna donde pondremos su nombre, con la etiqueta td. Además, recorremos el array $notasAlumno, guardando el nombre de la asignatura en la variable $nombreModulo y la nota en la variable $nota. Construimos con estos dos valores una nueva columna de nuestra table.

Una vez vistas todas las notas de un alumno, cerramos una fila de la tabla, y volvemos a iterar con los siguientes alumnos y notas.


Ejemplo de uso de arrays

# 4. Funciones

## 4.1. Introducción

Todas las aplicaciones a menudo realizan la misma tarea en diferentes puntos del programa. Por ejemplo, podríamos mostrar el logo de nuestra compañía en varias páginas web distintas o en diferentes partes del programa. El código sería el siguiente:

```
echo "<hr width='50' align='left'/>";
echo "<img src='/images/logo.jpg' width='50'
height='50'/>";
echo "<hr width='50' align='left'/><br>";
```

Las funciones nos permiten agrupar líneas de código bajo un nombre, de forma que no tengamos que repetir las mismas instrucciones una y otra vez siempre que queramos hacer lo mismo en nuestro programa. Podríamos hacer una función que contenga los enunciados anteriores y darle el nombre mostrar_logo. Entonces, cada vez que el programa necesite mostrar el logo, sólo tendremos que llamar a la función que contiene los enunciados de la siguiente forma:

```
mostrar_logo();
```

Obsérvense los paréntesis que hay detrás del nombre de la función. Son obligatorios, es la forma de indicar a PHP que se está llamando a una función.

## 4.2. Ventajas del uso de funciones

El uso de funciones ofrece las siguientes ventajas:

- Menos escritura: basta con escribir las instrucciones una vez, dentro de la función. Cuando queramos ejecutar dichas instrucciones, sólo tendremos que llamar a la función que las contiene.
- Lectura más fácil: es mucho más fácil de entender el nombre de una función que varias instrucciones. Además, tenemos que procurar poner nombres orientativos de lo que hace una función (como por ejemplo, mostrar_logo).
- Menos errores: después de haber escrito la función y haber arreglado sus problemas, funcionará correctamente cada vez que la usemos.
- Más fácil de cambiar: es muy frecuente querer cambiar la forma de hacer una tarea. Con el uso de funciones sólo hará falta cambiar esa tarea en un lugar. Si no usáramos funciones, tendríamos que mirar en todo el código en busca de los sitios donde se realiza esa tarea, y cambiar el código en cada lugar.

## 4.3. Creación de funciones

Para crear una función emplearemos la siguiente sintaxis:

```
function nombredelafuncion(){
     bloque de instrucciones;
     return;
}
```

Por ejemplo, para crear la función que nos muestre el logo de la empresa haríamos:

```
function mostrar_logo(){
     echo "<hr width='50' align='left'/>";
     echo"<img src='/images/logo.jpg' width='50'
height='50'/>";
     echo "<hr width='50' align='left'/><br>";
     return;
}
```

La sentencia return detiene la ejecución de la función y regresa al programa principal. No es obligatorio ponerlo, pero facilita la comprensión de la función. No obstante, habrá casos en que querremos que nuestra función devuelva algún resultado. Usaremos return con este fin. Esto lo veremos en el apartado Obtener un valor de una función.

La sentencia return no se pone siempre al final del bloque de instrucciones de una función. Puede usarse también en medio, normalmente dentro de bloques condicionales, en los que si se cumple una condición queremos salir de la función.

La creación de funciones se puede hacer en cualquier punto del programa, pero la práctica común es ponerlas todas juntas al inicio o al final del archivo del programa principal. Las funciones que se vayan a usar en más de un programa pueden estar en un archivo separado. Cada programa tiene acceso a las funciones del archivo externo. Veremos en otro tema cómo organizar nuestro programa en archivos y cómo acceder a ellos.

## 4.4. Uso de variables en funciones

En una función se pueden crear variables. A éstas se le conocen como variables locales, ya que al crearse en una función, pueden utilizarse dentro de ésta pero no fuera, no están disponibles en el programa principal. No obstante, si queremos crear una variable en una función y que podamos usarla en cualquier parte del programa, pondremos la palabra global delante del nombre de la variable cuando la creemos.

Por ejemplo, veamos el siguiente fragmento de código:

```
function suma() {
     $a = 2;
     $b = 3;
     $resultado = $a + $b;
}
suma();
echo $resultado;
```

Hemos creado una función suma, en ella se crean tres variables. A continuación, llamamos a la función y se produce el output de la variable $resultado. Pero no se generará ningún output, la variable $resultado no tiene valor. En teoría debería valer 5, ¿no? En efecto, este es el valor de la variable $resultado de dentro de la función, pero fuera hemos creado otra variable $resultado, sin valor.

Para solventar este problema, creamos la variable $resultado dentro de la función como global:

```
function suma() {
     $a = 2;
     $b = 3;
     global $resultado = $a + $b;
}
suma();
echo $resultado;
```

Ahora sí obtendríamos el output deseado. Obviamente, debemos especificar que la variable es global antes de usarla. Si intentáramos hacer el echo antes de la llamada a la función, o antes de crear la variable, no obtendríamos ningún output.

## 4.5. Paso de valores a una función

Podemos pasar valores del programa principal a una función cuando la llamamos, poniendo entre paréntesis los valores que queremos pasar:

```
nombredefuncion (valor1, valor2, ...);
```

Pero para hacer esto, tenemos que preparar a la función para que sea capaz de recibir valores. Al crear la función, especificaremos las variables donde se van a almacenar cada uno de los valores que la función recibirá:

```
function nombredelafuncion($variable1, $variable2, ...){
     bloque de instrucciones;
```

```
        return;
    }
```

Por ejemplo, vamos a retocar la función anterior suma para que reciba dos valores del programa principal:

```
function suma($a, $b) {
    global $resultado = $a + $b;
}
suma(2, 3);
echo $resultado;
```

Se puede ver que ahora la declaración de las variables $a y $b de la función suma se hace en la cabecera, entre los paréntesis. En la llamada a la función es donde le damos valor a esas dos variables. Es importante el orden en que ponemos los valores en la llamada: el primer valor se copia en la primera variable, el segundo valor en la segunda variable…

Al crear las funciones podemos especificarle tantas variables como queramos. Claro está, luego en la llamada tendremos que darle valor a cada una de estas variables. Los valores pueden ser variables o literales. Las siguientes llamadas a la función suma serían correctas:

```
suma(2*4, 3+5);
suma(2, $valor);
suma($a, $b);
```

Si en la llamada a una función no enviamos suficientes valores, la función establece el valor faltante en una cadena vacía para una variable en cadena, o a 0 para un número. Si enviamos demasiados valores, la función ignora los valores adicionales.

Además de lo dicho en el párrafo anterior, en función del nivel de mensajes de error en el cual tengamos configurado PHP, podremos recibir un mensaje de advertencia.

También podemos pasar los valores mediante un array. Por ejemplo:

```
function sumanumeros($numeros) {
    global $suma;
    foreach ($numeros as $valor){
        $suma = $suma + $valor;
    }
}
$array = array(1, 2, 3, 4, 5);
sumanumeros($array);
echo $suma;
```

Podemos fijar valores predeterminados para usarse cuando un valor no se pase. Los valores predeterminados se establecen cuando escribimos la función asignando un valor predefinido para el valor o los valores que estemos esperando, como sigue:

```
function suma($a=2, $b=3) {
    global $resultado = $a + $b;
}
```

Si uno de los valores no se pasa (o si no se pasa ninguno), la función usa los valores predefinidos asignados. Pero si se pasa un valor, éste se usa en lugar del valor predeterminado. Por ejemplo, podríamos realizar las siguientes llamadas:

```
suma(5, 8);
suma(6);
suma();
```

Los resultados son, en orden consecutivo: 13, 9, 5.

## 4.6. Paso de valores de una función

Cuando llamamos a una función, hemos visto que podemos pasarle valores. Pero la función también puede generar distintos valores que queramos utilizar en el programa principal. Usaremos el enunciado return para devolver un valor al programa que ha hecho la llamada a la función.

Veamos un ejemplo:

```
function suma($a=2, $b=3) {
     $resultado = $a + $b;
     return $resultado;
}
$resultado = suma(9,4);
```

En este código la función suma devuelve el valor de la variable $resultado (return $resultado). Y la llamada a la función se realiza a la derecha del operador de asignación (=). Con esto, asignamos el valor que devuelve la función a una variable, que también se llama $resultado, declarada en el programa principal. Hay que tener claro que hay dos variables que se llaman igual, $resultado, pero que son diferentes: una es local de la función, y la otra es del programa principal. En la llamada a la función hacemos que el valor de la variable local de la función se copie en la variable del programa principal.

Hay que tener en cuenta que el return sólo puede devolver un valor. No obstante, si queremos que una función devuelva varios valores, podemos usar un array para almacenarlos.

Podemos usar enunciados return en un bloque condicional para devolver diferentes valores para diferentes condiciones. Por ejemplo, la siguiente función devuelve dos posibles valores:

```
function comparar_valores ($a, $b){
     if ($a < $b){
          return "a es menor que b";
     }
     else{
          return "a no es menor que b";
     }
}
```

## 4.7. Ámbito de las variables

El ámbito de una variable se refiere al fragmento de código donde la variable ha sido declarada y puede ser usada. No es lo mismo declarar una variable al inicio del script, que hacerlo dentro de una función.

Una variable declarada en el script principal puede ser usada en ese mismo script, y no en funciones. A estas variables se les conoce como globales.

```
$variable_global = "Hola";
function f1 (){
     echo $variable_global;
     /*La instrucción anterior dará error. La variable no
     se puede usar en el ámbito de la función*/
}
```

Para trabajar con una variable global dentro de una función, deberemos pasar su valor como parámetro en la llamada a la función. De esta forma, podremos usar el valor de la variable global dentro de la función, pero las posibles modificaciones que hagamos sólo tendrán efecto dentro de la función.

```
$variable_global = "Hola";
function f1 ($variable_funcion){
     $variable_funcion = $variable_funcion." qué tal";
     echo $variable_funcion;
}
f1($variable_global); //Imprimirá "Hola qué tal"
echo $variable_global;//Imprimirá "Hola"
```

Las variables declaradas dentro de una función sólo pueden usarse dentro de la misma.

```
function f1(){
     $variable_funcion = 3;
}
echo $variable_funcion; /*ERROR. La variable no está
                          definida*/
```

Hay una manera de hacer que una variable declarada en una función pueda ser utilizada fuera de ésta, y es usando la palabra `global`.

```
function f1(){
     global $variable_funcion = 3;
}
echo $variable_funcion; //Imprimirá el valor 3
```

De igual forma, podemos usar variables globales dentro de funciones, usando el array `$GLOBALS`.

```
$a = "Hola";
function f1(){
     echo $GLOBALS['a']; //Imprimirá 'Hola'
}
```

### 4.8. Funciones incorporadas

En nuestros programas no vamos a tener que hacer todo el trabajo nosotros: PHP nos provee de una colección de funciones ya implementadas que nosotros sólo tendremos que llamar.

Por ejemplo, hemos visto ya funciones como unset, echo, exit, extract, sort, y veremos otras, como las que se encargan de interactuar con una base de datos MySQL.

## 5. Uso de formularios

### 5.1. Introducción

La gran mayoría de aplicaciones web, por no decir todas, están diseñadas para formular preguntas al usuario. Estas aplicaciones recogen la información de introducen los usuarios y la usan con diversos propósitos, como pueden ser guardar los datos en una base de datos o para realizar una comprobación en una instrucción condicional.

Los datos se recogen mediante los formularios HTML, mediante los cuales el usuario puede introducir de diversas formas los datos (ya sea con campos de texto, listas desplegables, cajas de selección, etc.).

En este capítulo se verá las herramientas que ofrece PHP para poder tratar los datos que introduce el usuario en un formulario.

### 5.2. Cómo recoge PHP los datos de un formulario

Cuando el usuario pulsa el botón Submit del formulario, se ejecuta la acción que contiene el parámetro action del formulario. En este parámetro especificamos el programa PHP que queremos ejecutar. Las instrucciones de este programa serán capaces de obtener los datos del formulario y operar con ellos. Hay diversas formas de acceder a estos datos:

| Función | Descripción |
|---|---|
| $_POST | Array que contiene los datos de cada campo del formulario, siempre que éste tenga method=”POST” |
| $_GET | Array que contiene los datos de cada campo del formulario, siempre que éste tenga method=”GET” |
| $_REQUEST | Array que contiene los datos de $_POST, $_GET y $_COOKIE |

Supongamos que tenemos un formulario cuyo método está definido a POST (method="POST"), y tiene los campos (inputs) "nombre" y "edad". Una forma de obtener los datos de estos campos sería con las siguientes instrucciones:

```
$nombre = $_POST["nombre"];
$edad = $_POST["edad"];
```

Podemos observar que accedemos a los campos del formulario mediante el array $_POST usando su nombre.

El siguiente ejemplo muestra cómo listar todos los campos de un formulario junto con el valor introducido por el usuario en cada uno:

```
foreach ($_POST as $campo => $valor){
     echo "$campo = $valor<br/>";
}
```

## 5.3. Post VS Get

Para enviar información de un formulario, se usa uno de los dos métodos. Las diferencias entre uno y otro son:

- Método GET: los datos del formulario se pasan agregándolos a la URL que ejecuta la acción del formulario. Por ejemplo:

  procesaform.php?nombre=Dani&apellido=Santiago

  Las ventajas de este método son su simplicidad y velocidad. Las desventajas son que se pueden pasar menos datos y que la información se puede ver en el explorador, lo cual es un problema de seguridad.

- Método POST: los datos del formulario se pasan como un paquete en una comunicación separada con el programa de procesamiento. Las ventajas de este método son que se puede pasar información ilimitada y los datos están más seguros. La desventaja es que es un método más lento.

## 5.4. Obtener datos de un formulario

Como hemos visto, PHP nos proporciona los arrays $_GET y $_POST donde se guardan los valores que ha introducido el usuario en los campos de nuestros formularios. Recordemos que $_GET se utiliza para formularios de tipo "get", y $_POST para formularios de tipo "post".

Así pues, para coger el valor de un campo de texto llamado "nombre", de un formulario de tipo "post", escribiremos:

```
$var = $_POST["nombre"];
```

Sabemos que los formularios pueden tener diversos tipos de inputs: campos de texto, listas desplegables, botones de selección, casillas… El proceso de recoger los valores de cada input es muy similar, pero existen casos especiales, como las casillas, ya que el usuario puede seleccionar varias de ellas a la vez. Veamos un ejemplo:

En el siguiente formulario, el usuario puede seleccionar varias aficiones, representadas con una lista de casillas. Vamos a almacenar los datos de esta lista de casillas en un array:

```
echo "<input type='checkbox' name='aficiones[leer]'
value='Leer'>Leer<br/>";
echo "<input type='checkbox' name='aficiones[deporte]'
value='Deporte'>Practicar deporte<br/>";
echo "<input type='checkbox' name='aficiones[programar]'
value='Programar'>Programar<br/>";
```

Podemos acceder a los datos de la siguiente forma:

```
$_POST[aficiones][leer];
$_POST[aficiones][deporte];
$_POST[aficiones][programar];
```

Otra forma de acceder a los datos que ha introducido el usuario en un formulario es el uso de la estructura foreach, que es útil para recorrer un array. Recordemos que los datos de un formulario se almacenan en el array $_POST (o $_GET, en función del tipo de formulario). Veamos un ejemplo, en el que se comprueba que todos los campos del formulario tienen algún valor:

```
foreach($_POST as $valor){
     if($valor==""){
          echo "Debes completar todos los campos";
          mostrar_formulario();
          exit();
     }
}
echo "Bienvenido";
```

En el código hemos introducido un foreach que recorre cada posición del array $_POST. Si encuentra alguno vacío, es decir, igual a una cadena de caracteres vacía (""), muestra un mensaje y llama a una función que crea el formulario para que el usuario vuelva a introducir sus datos. La función exit() es importante: hace que el programa se detenga. Tiene sentido ya que si hay algún campo vacío, se debe volver a mostrar el formulario y ya está. Si no pusiéramos la función exit(), el programa seguiría ejecutándose y se mostraría el mensaje de bienvenida que hay después del foreach.

Veamos ahora un ejemplo en el que recorremos los campos de un formulario para asegurarnos de que el usuario los ha rellenado, todos menos un par, llamados "segundoTelefono" y "fax":

```
foreach($_POST as $campo => $valor){
     if($campo != "segundoTelefono" and $campo != "fax"){
          if($valor==""){
               echo "Debes completar todos los campos";
               mostrar_formulario();
               exit();
          }
     }
}
echo "Bienvenido";
```

## 5.5. Verificar el contenido de los campos de un formulario

Una práctica útil en nuestras aplicaciones web es asegurarnos de que la información que introduce un usuario en los campos de los formularios es correcta: por ejemplo, los números de teléfono no contiene letras, no hay números en los nombres, etc. También podemos proteger a la aplicación o la base de datos de ataques de usuarios que introduzcan scripts en los campos de los formularios.

Esta verificación de contenido la podemos hacer usando expresiones regulares. Veamos un ejemplo:

```
if(!ereg("[A-Za-z' -]{1,50}", $apellido)){
     echo "Ha introducido caracteres erróneos en el
apellido";
     mostrar_formulario();
     exit();
}
echo "Bienvenido";
```

En el código anterior comprobamos mediante la función ereg que el valor de la variable $apellido sólo contenga letras (mayúsculas o minúsculas), espacios, comilla simple o guión.

## 5.6. Formulario con más de un botón

En ocasiones nuestras aplicaciones tendrán formularios con más de un botón. Por ejemplo, en un carrito de la compra, tendremos un botón para confirmar el pedido y otro para cancelarlo. Ambos botones estarán dentro del mismo formulario, el cual sólo puede realizar una acción en su atributo action. Necesitamos una manera de saber qué botón ha pulsado el usuario para luego realizar las operaciones que queramos asociadas a cada botón. Veamos como hacer esto en PHP.

Tenemos este formulario:

```
<form action="acciones.php" method="POST">
     <input type="text" name="nombre"/><br/>
     <input type="submit" name="boton"
     value="Aceptar"/><br/>
     <input type="submit" name="boton"
     value="Cancelar"/><br/>
</form>
```

Los dos botones deben llamarse igual (tienen el mismo valor en name).

La página acciones.php tendrá el siguiente código:

```
if($_POST["boton"] == "Aceptar"){
     mostrar_pagina1();
}
else{
     mostrar_pagina2();
}
```

Para saber el botón que ha pulsado el usuario utilizamos el array $_POST y nos fijamos en el valor de la variable asociada a los botones del formulario (boton).


Ejemplo de uso de formularios

## Test de autoevaluación

¿Cuál de los siguientes no es un bucle reconocido por PHP?

a) While (condición) {}
b) Do {} while(condición);
c) Do while (condición) {}
d) for([contador=valorInicial];[condición];[incremento]){}

¿Con qué función se puede saber el número de elementos que hay en un array en PHP?

a) size()
b) length()
c) LBound()
d) count()

¿Qué tipo de variable es la variable $_POST?

a) Es un array
b) Es una cadena de caracteres
c) Es un entero
d) Depende de cómo la declare el programador

¿Cómo puedo añadir un nuevo elemento a la posición siguiente del array?

a) No se puede hacer esto, los arrays tienen un tamaño fijo predefinido
b) $miArray [$posicion] = $valor
c) $miArray [] = $valor
d) $miArray += $valor

## Recursos y enlaces

- Manual oficial de PHP: estructuras de control
- Manual oficial de PHP: arrays
- Manual oficial de PHP: funciones
- Manual oficial de PHP: formularios

## Conceptos clave

- **Construcciones condicionales**: son aquellas que ejecutan un bloque de instrucciones únicamente si se cumplen ciertas condiciones.
- **Construcciones iterativas**: establecen un bloque de instrucciones que se repiten.
- **Arrays:** son variables que almacenan un grupo de valores bajo un único nombre de variable.
- **Funciones:** las funciones nos permiten agrupar líneas de código bajo un nombre, de forma que no tengamos que repetir las mismas instrucciones una y otra vez siempre que queramos hacer lo mismo en nuestro programa.
- **Formularios:** son herramientas que ofrece el lenguaje HTML para recoger los datos que el usuario introduzca.

# Ponlo en práctica

## Actividad 1

Crea una aplicación web usando PHP que muestre una página con una tabla de 5x5. Las casillas contiguas irán alternando su color de fondo entre gris y blanco. El contenido de cada casilla será un número, teniendo el 1 la primera casilla, el 2 la siguiente, y así sucesivamente. Se debe hacer uso de construcciones condicionales e iterativas para construir la tabla.

SOLUCIÓN

## Actividad 2

Crea una aplicación web usando PHP que tenga un array de tres posiciones con el siguiente texto en cada posición: hola, cómo, estás. A continuación usa un bucle para recorrer el array y mostrar su contenido.

SOLUCIÓN

## Actividad 3

Crea una aplicación web usando PHP en la que se declare una función suma con dos parámetros, a y b, éste último con el valor 24 por defecto. La función sumará el valor de los dos parámetros y devolverá el resultado. Haz una llamada a la función suma pasándole dos valores, y otra pasándole sólo uno. ¿Qué devolverá cada llamada?

SOLUCIÓN

## Actividad 4

Crea una aplicación web usando PHP que conste de las siguientes páginas:

- La primera contendrá un formulario en HTML que mostrará un campo de texto donde el usuario podrá escribir su nombre, y un botón de tipo submit. El formulario deberá ser de tipo POST.
- La segunda mostrará el texto “BIENVENID@” acompañado del nombre que haya escrito el usuario en la página anterior.

SOLUCIÓN