# Los bucles for, while y foreach

Los bucles son estructuras que controlan el flujo de ejecución del código, pero ya no en forma "condicional" o "selectiva" como en el caso de los if/else y switch, sino que determinan la ejecución "repetitiva" de un código.

Debido a su característica repetitiva, son ideales para recorrer vectores, o conjuntos de filas o registros traídos de una base de datos.

Existen tres funciones básicas en PHP (no son las únicas) para las estructuras de tipo bucle: **for**, **while** y **foreach**. Veamos cada una de ellas, y ejemplos de su uso.

### Cómo repetir una acción una cantidad fija de veces: el bucle "for".

Imaginemos que precisamos hacer una simple lista. Supongamos que tenemos que listar los 31 días de un mes, y cada línea debe decir "Hoy es el día **X**", o sea, la primera línea debe decir "Hoy es el día 1", la siguiente "Hoy es el día 2" y así sucesivamente.

En vez de escribir 31 líneas de código con 31 echos o prints, uno por cada frase, podemos lograr idéntico resultado mediante un bucle "for", en solo 3 líneas de código.

Este tipo de instrucción llamada **for** permite **ejecutar una orden "a repetición" un número "exacto" de veces**, es decir, el número de repeticiones debe ser conocido con anterioridad (en este caso, conocemos la cantidad de días del mes, 31).

El bucle **for** trabaja modificando el valor de una variable que utiliza como "**índice**" o indicador del **número de vuelta** en el que se encuentra el bucle (un "cuentavueltas"); por ejemplo:

for (**$indice=1**; etc...

esta variable $indice debe ir en primer lugar, como primer parámetro dentro de los paréntesis del for, fijando el valor original que tendrá esa variable al **inicio** del bucle.

Luego, como segundo argumento, se fija el valor **tope** (o final) de esa variable -es el número hasta el cual se repetirán las instrucciones-;

for ($indice=1; **$indice<10**; etc...

y en tercer lugar, se especifica qué **incremento** (o decremento) sufrirá esa variable luego de terminar de dar **cada vuelta** del bucle:

for ($indice=1; $indice<10; **$indice=$indice + 1**) {

*Aquí van las instrucciones que ejecutará 10 veces (mientras $indice valga menos que 10);*

}

Veamos un ejemplo "abstracto" de la estructura de un bucle "for":

---

```
for (valorinicial; valorfinal; incremento){

   bloque a ejecutar;
```

```
}
```

---

Y ahora un caso real. Crearemos el archivo **buclefor.php**:

---

```
<?php
for ($dia=1; $dia<32; $dia=$dia+1){

    print("Hoy es el día ".$dia."<br />");

}
?>
```

---

**Análisis "paso a paso" del ejemplo anterior:**

Al entrar al bucle "for", $dia se inicializa (toma un valor inicial) de "1".

A continuación, se verifica si ya se cumplió la condición de permanencia dentro del bucle (que $dia sea menor a 32). Como aún es verdad esto, se ejecuta el bloque de sentencias encerrado entre las llaves del "for", y luego de ejecutarlas, se incrementará $dia en "1" más, es decir, el valor de $dia se convertirá en "2".

Al ejecutarse el bloque de sentencias, se reemplazará la variable $dia por su valor actual (en esta primera vuelta, un "1"), por lo que se imprimirá "Hoy es el día 1<br />".

En la segunda vuelta, $dia ya vale "2", por lo que se imprimirá "Hoy es el día 2<br />", y así sucesivamente, hasta que se haya impreso "Hoy es el día 31"; en ese instante, $dia valdrá 31, y al volver a pasar por el bucle, tomará el valor 32, con lo que ya no será "menor a 32" que era la condición para seguir, y se saldrá del bucle, siguiendo con la ejecución del código que venga a continuación del final del for (abajo de la llave final de cierre).

Este tipo de bucles es el más usado para recorrer los **vectores de índices numéricos secuenciales (consecutivos)**, ya que mediante la función **count** recientemente vista, podemos conocer la cantidad de vueltas que debe dar el bucle hasta recorrer todas las posiciones del vector.

Otro ejemplo: recorrer y mostrar un vector cargado con productos:

---

```
<?php

$productos = array(1 => "manzanas","naranjas","peras","uvas");

$cuantos = count($productos);

for ($i=1; $i<=$cuantos; $i=$i+1){

    print("El producto ".$i." es: ". $productos[$i]."<br />");

}
?>
```

Esto mostrará:

El producto 1 es: manzanas
El producto 2 es: naranjas
El producto 3 es: peras
El producto 4 es: uvas

**Cómo enviar un vector por POST:**

En PHP podemos aprovechar la facilidad de enviar múltiples datos en **un solo vector** de una página a otra, en vez de enviar muchas variables (entre otras ventajas, esto **facilita la validación** de los datos enviados en el servidor).

Veamos cómo hacer un formulario que en lugar de enviar múltiples variables envíe **un único vector**:

Ejemplo: formulario en el archivo **creavector.htm**:

```
<form action="bucle.php" method="POST">
Nombre: <input type="text" name="datos[]"><br>
Edad: <input type="text" name="datos[]"><br>
Estado civil: <input type="text" name="datos[]"><br>
Sueldo: <input type="text" name="datos[]"><br>
<input type="submit" value="Enviar">
</form>
```

Notemos cómo el atributo "name" de cada input es el mismo, siempre seteando un vector llamado "datos", y en este caso, sin aclarar números de índices (por lo que PHP asignará al primer dato un cero, al segundo un 1 y así sucesivamente).

Este formulario envía los datos del vector a bucle.php:

Archivo **bucle.php**:

```
<?php

$cuantos = count($_POST["datos"]);

for ($i=0; $i<$cuantos; $i=$i+1){

    print("Dato ".$i.": ".$_POST["datos"][$i]."<br>");

}
?>
```

**Formularios armados con un bucle "for".**

Ahora vamos a preparar dos formularios que utilizaremos en la lección de hoy.

Veremos que incluso el formulario de carga de datos será armado mediante un bucle -for, en este caso- (por eso la extensión ".php" de este archivo ahora es necesaria).

El primero de los archivos creará tres campos donde ingresar datos, y el segundo archivo, seis (solo a efectos de observar cómo generamos formularios distintos con solo cambiar "un" número, y sin cambiar nada en el archivo que procesará esos formularios).

Ambos apuntan a una misma página "whilebucle.php":

Archivo **formu1.php**:

---

```
<form action="whilebucle.php" method="POST">
<?php
for ($i=0; $i<3; $i=$i+1){

    print("Número ".$i.": <input type=\"text\" name=\"numeros[]\"><br />");

}
?>
<input type="submit" value="Enviar">
</form>
```

---

Otra variante, ahora con 6 inputs: archivo **formu2.php**:

---

```
<form action="whilebucle.php" method="POST">
<?php
for ($i=0; $i<6; $i=$i+1){

    print("Número ".$i.": <input type=\"text\" name=\"numeros[]\"><br />");

}
?>
<input type="submit" value="Enviar">
</form>
```

---

De ahora en más, podemos utilizar estos inputs como cualquier otro input, pero con la facilidad de validar los datos mediante un bucle (ya veremos más adelante cómo validar).

**Cómo repetir algo una cantidad desconocida de veces: el "while".**

Muchas veces nos sucederá que debemos realizar una acción hasta que pase algo, pero no sabemos a ciencia cierta cuándo sucederá ese algo.

Por ejemplo, los dos formularios distintos del ejercicio anterior apuntan a una misma página que procesa los datos; en uno de los formularios, hay solamente tres campos, con lo cual el vector generado será de tres celdas; en el otro formulario, hay seis campos, con lo cual el vector generado será de seis posiciones.

La página que recibe los datos ("whilebucle.php") no sabe a priori cuántas celdas tiene el vector, y si bien podríamos utilizar la función **count** para saber su tamaño y recorrerlo con un **for**, esta vez lo haremos mediante un **while**.

De paso, veremos la forma de recorrer un vector cuando no sabemos cuántas celdas tiene, o cuando sus índices no son numéricos, o cuando son numéricos pero no siguen un orden secuencial.

Archivo **whilebucle.php**:

---

```
<?php
while (list($clave,$valor) = each($_POST["numeros"])){

    print("El elemento ".$clave." contiene un ".$valor."<br />");

}
?>
```

---

**Las funciones "list" y "each".**

Hemos utilizado dentro del bucle while dos funciones nuevas: **list** y **each**.

La función **each** "captura" el elemento del vector actual, aquel al cual el puntero o índice interno que todo vector posee está apuntando en ese momento, y además de traer ese elemento (recordemos que un elemento de un vector es un par "índice-valor"), desplaza el "puntero" al siguiente elemento del vector.

Cuando se ejecuta la función each, a su izquierda debe precederla una variable donde quedará almacenado todo el elemento capturado por each; la función each se encarga de convertir a esa variable ("$este" en este caso) en un vector, con cuatro posiciones:

Ejemplo de **each**:

---

```
<?php
$datos[nom] = "Juan Pérez";
$datos[ed] = 24;
$datos[est] = "casado";
$datos[suel] = 800;
$este = each($datos);
/*Convierte a este en un vector de 4 celdas
Para visualizarlo, imprimiremos las 4 celdas de $este:*/
print("El índice de este elemento es ".$este[0]."<br />");
print("El valor de este elemento es ".$este[1]."<br />");
print("El índice de este elemento es ".$este[key]."<br />");
print("El valor de este elemento es ".$este[value]."<br />");
?>
```

---

Notemos que "$este" fue convertido por obra de "each" en un vector de **cuatro** elementos: el índice del primero es "0" y el del segundo "1", y contienen, respectivamente, el que era el índice del elemento del vector $datos que se trajo el each ("nom", en este ejemplo), y el valor de ese mismo elemento del vector

$datos ("Juan Pérez", en este caso). Luego, el tercer y cuarto elementos de $este vuelven a contener la misma información, pero con los índices "key" y "value" (clave -o índice-, y valor).

En el ejercicio anterior, en el archivo "whilebucle.php", para ganar en claridad, hemos simplificado y utilizamos solo las primeras dos posiciones que crea each: en la primera almacenamos el índice y en la segunda el valor de ese elemento seleccionado. Pero en vez de adjudicarle esos dos datos a una variable/vector (como $este) se los adjudicamos mediante la función **list a dos variables** que van a recibir los datos: list($clave,$valor).

En list($clave) se guarda la clave del elemento traído por each, y en list($valor) el valor.

El bucle while lo hacemos entonces "mientras" el list($clave,$valor) sea igual al each de $numeros. Es decir, mientras se pueda producir esa asignación de each hacia list.

Copiamos el código nuevamente aquí para examinarlo otra vez:

Archivo **whilebucle.php**:

---

```
<?php
while (list($clave,$valor) = each($_POST["numeros"])){

    print("El elemento ".$clave." contiene un ".$valor."<br />");

}
?>
```

---

Las variables $clave y $valor en cada vuelta del bucle contienen el índice y el valor del elemento que each se trajo.

Se sale del bucle cuando sea falsa esa asignación propuesta (que el valor que captura each se pueda pasar a list...) es decir, cuando se haya recorrido hasta el último elemento al vector $numeros y ya no haya un elemento que se traiga each.

Este método que hemos aprendido es el más utilizado para recorrer vectores de índices no numéricos, o numéricos no consecutivos.

**Cómo repetir algo una cantidad desconocida de veces con "foreach".**
Para simplificar el recorrido de vectores, existe una estructura llamada foreach, que sirve exclusivamente para recorrer vectores (si se le pasa como parámetro algo que no sea un vector generará un error).

Su estructura "mínima" (la que aprenderemos por ahora) es la siguiente:

**foreach ($vector as $valor) {**

**bloque a ejecutar**

**}**

En cada vuelta del bucle, el valor del elemento actual del vector se asigna a la variable *$valor* y el índice del vector se incrementa en uno (así en la siguiente vuelta, se estará parado en el elemento siguiente).

Veamos el anterior ejemplo, pero simplificado gracias a foreach:

Archivo **whilebucle.php**:

---

```php
<?php
foreach ($_POST["numeros"] as $valor){

    print("El elemento contiene un ".$valor."<br />");

}
?>
```

---

Y si quisiéramos mostrar los índices:

---

```php
<?php
foreach ($_POST["numeros"] as $clave => $valor){

    print("El elemento: ".$clave." vale: ".$valor."<br />");

}
?>
```

## Las funciones

**Qué son y para qué sirven las funciones.**
Por ejemplo, imaginemos que ejecutamos esto:

**mail ($destino, $asunto, $mensaje);**

"En alguna otra parte" algún programador **"definió" un conjunto de instrucciones a las que llamó "función mail",** y determinó que al llamarla deberíamos pasarle 3 datos (esos datos se llaman "parametros" o "argumentos" de las funciones). En este caso, los 3 datos son el mail de destino, el asunto y el cuerpo del mensaje (en ese orden, así lo decidió quien creó la función mail).

También decidió que podríamos pasarle opcionalmente un cuarto dato, con cabeceras (headers) de los mails, tales como el From, Reply-to, CC, etc. Y determinó que si estaban esos datos, se conectaba con un servidor SMTP, verificaba que hubiese sido enviado el mensaje, y devolvía "verdadero" (TRUE) o FALSE en caso contrario.

De todo lo cual podemos deducir que **las funciones primero se "declaran" definiendo qué harán, y luego se "llaman" o "ejecutan"** (se usan!).

Las funciones poseen una potencia muy grande, ya que cada una de ellas "sabe" **cómo hacer algo con los datos que le pasamos**: la función print, por ejemplo, sabe escribir en el código HTML de una página antes de mandarla a nuestro navegador; la función mail, sabe mandar emails. Son capaces de "hacer" cosas, simplemente recibiendo la cantidad de **parámetros** o **argumentos** necesarios (así se llaman los datos que recibe una función cuando es llamada). Y aunque la función fue creada una sola vez, es reusable infinitas veces, en todo nuestro sitio y en otros proyectos.

Pero como si fuera poco con las más de 1.200 funciones predefinidas que trae PHP de fábrica, nosotros también podemos crear **nuestras propias funciones**, todas las que querramos. Por ejemplo, podríamos crear una función para validar datos de un formulario, podríamos crear una función para mostrar resultados de una base de datos, podríamos crear una función para convertir texto de minúscula a mayúscula, etc.

Vamos a empezar viendo **cómo se declara una función** (como se "define" lo que va a ser capaz de hacer). Esto se logra escribiendo la palabra "function", luego el "nombre" que le ponemos a esa función, y

luego entre paréntesis los "argumentos" o datos que va a recibir, y dentro de un par de llaves, el bloque de sentencias que va a ejecutar:

*function nombre (argumentos) {*
*bloque a ejecutar*
*}*

**Nuestra primera función.**

Manos a la obra. Crearemos el archivo **funcion1.php**:

```
<?php
function mostrar($vector){

while (list($clave,$valor) = each($vector)){
print ("El elemento ".$clave." vale: <b>".$valor."</b><br />");
}

}
?>
```

Esta función así declarada nos permitirá imprimir cualquier vector que le pasemos como argumento al momento de "llamar" a la función. "Llamar" a la función significa ejecutarla. Deben pasársele la misma cantidad de argumentos que los especificados en la declaración de la función (en este ejemplo, uno solo, el nombre del vector a imprimir).

Así se llamaría a esta función:

```
<?php
//primero cargamos un par de vectores:
$propiedades = array("casas", "departamentos", "campos", "terrenos", "quintas");

$vehiculos = array("autos", "motos", "camiones", "cuatriciclos", "bicicletas");

//ahora llamamos a la función para que los muestre:
mostrar($propiedades);//esto llama a la función mostrar

print("<br />");

mostrar($vehiculos); //esto llama a la función mostrar otra vez
?>
```

Ambos bloques del código anterior deben estar en el mismo archivo).

**Pasar parámetros por valor o por referencia.**

Nuestra función recibe un valor que le pasamos cuando llamamos a la función: en este ejemplo, recibe primero al vector "propiedades" y luego al vector "vehículos". Como en la declaración de la función al lado del nombre de la misma, entre paréntesis, pusimos la variable $vector, ésta variable (o vector) es el que va a manipular dentro de la función al vector pasado como argumento. **Dentro de la función no llamamos al vector que vino de fuera con su verdadero nombre (ya que podríamos no saberlo, y es así**

**casi siempre) sino con un "alias" o sobrenombre mediante el cual lo manejamos dentro de la función.**

Vamos a ver un ejemplo donde se vea más claramente:

```php
<?php

function cuadrado($numero){
return $numero * $numero;
}

$num = 6;
$resultado = cuadrado($num);
print($num." al cuadrado da: ".$resultado."<br />");
//imprime 36
?>
```

Si a continuación imprimimos $num, **no cambió su valor**, sigue siendo 6. A esto se llama pasarle argumentos "por valor" a una función: la función no trabaja sobre la variable original, sino que crea **una copia** del valor, una copia de la variable original con otro nombre: un "alias" (en este ejemplo, $numero).

Pero también está la posibilidad de que querramos que la función **sí modifique el valor de la variable original.** A eso se le llama pasarle un argumento "por referencia", es decir haciendo referencia a la variable original y no a una copia o alias.

Para pasar por referencia un argumento, se antepone un **ampersand (&)** a la variable que queremos que modifique su valor por referencia, o directamente anteponiendo el ampersand en la declaración de la función (en el primer caso solo actúa "por referencia" en esa específica llamada a la función, y en el segundo caso, siempre).

Veamos el caso de que siempre reciba por referencia:

```php
<?php

function cuadrado(&$numero){//va a tomar siempre por referencia los datos que le pasen
$numero = $numero * $numero;
return $numero;
}

$num = 6;
print("La variable \$num vale: ".$num."<br />");
print("\$num al cuadrado da: ");
$resultado = cuadrado($num);
print($resultado."<br />");
print("Nuevo valor de \$num: ".$num."<br />");//modificó su valor!!
?>
```

Notemos varias cosas:

1) La palabra **"return"** sirve para **devolver un resultado** luego de la ejecución de la función. Se lo pasa nuevamente a la variable original, en este caso $num.

2) Los signos $ de los nombres de variables dentro de un print deben "escaparse", esto es, que no se ejecuten si uno lo que quiere es que se vea en la pantalla el texto $num y no el contenido de la variable $num.

3) Al haber un ampersand delante del argumento en la declaración de la función, todas las veces que le pasemos un dato la variable original verá modificado su valor.

**Includes**

Existe una función llamada *include* muy útil que lo que nos permite es **incluir el código** de un documento en otro (ya sea PHP o HTML). No se confundan, *include* incluye *código* y no por ejemplo, imágenes o archivos swf.

Sintaxis:

```
<?php
include("pagina.php");
?>
```

Utilidades:

- En cuanto a código PHP, son muy útiles para por ejemplo tener un único archivo con todas las funciones que se usarán en todas las páginas del sitio (sobre todo si son funciones que se usan reiteradas veces y en distintas páginas). Para usar las funciones en ese archivo "biblioteca" usamos la función *include* para incluirlo en la página donde lo necesitemos.
- En cuanto a código HTML, nos sirve para incluir DIVs enteros. Supongamos que tenemos un sitio medianamente grande de 50 páginas individuales, cada una con un menú a la izquierda. Sería muy práctico que todas estas páginas incluyeran al DIV del menú con un include, para que el día de mañana al tener que agregar o quitar un link el menú sólo tengamos que modificar el archivo que lo contenga y no a las 50 páginas individualmente.

Ejemplo:

encabezado.htm

```
<div id="encabezado">
Bienvenido a mi sitio<
</div>
```

biblioteca.php

```
<?php
print date("d/m/y H:i a");
?>
```

index.php

```
<?php include("encabezado.htm");
<?php include("biblioteca.php");
echo FechaConFormato(); ?>
```

El resultado por pantalla sería algo como:

```
Bienvenido a mi sitio
Hoy es 04 de Nov 16:30 hs.
```