Programación Web con PHP y MySQL– Nivel 2

Unidad 6 (parte 2): Vectores

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Indice

Unidad 6 (parte 2): Vectores

Objetivos

Que el alumno logre:

- Comprender la utilización de las variables con más de un valor.

Tablas, matrices, vectores o arrays

Vamos a ver varios ejemplos de trabajo con arrays (vamos a poder encontrarlos también como arreglos, vectores, matrices o tablas en castellano) en PHP que ilustrarán un poco el funcionamiento de algunas de las funciones de arrays más populares que trae consigo PHP.

Los arreglos pueden ser de una o más dimensiones, los de una dimensión, son llamados comúnmente "vectores".

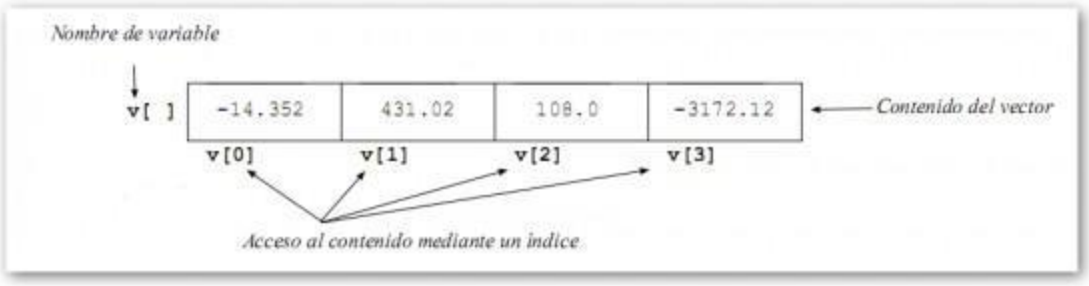
A diferencia de otros lenguajes, en PHP, un vector puede tener elementos de distintos tipos.

ARRAYS INDEXADOS

Las variables de tipo Array son como variables con muchos "**compartimientos**" que pueden almacenar varios valores, a los que se puede acceder mediante un índice. Es decir, un **array** es una variable que está compuesta de varios elementos, cada uno de ellos catalogado dentro de ella misma por medio de una clave.

Dijimos entonces que para hacer referencia a un elemento del vector, se utiliza un índice, que indica la dirección en donde se encuentra un determinado valor. El índice en un arreglo comienza siempre por cero. (Más adelante veremos que el índice de un vector, no necesariamente debe ser un número entero, sino que también puede ser un texto).

En la sintaxis de los Arrays en PHP, el índice (o clave) se indica entre corchetes.



Hay varias maneras de definir un array:

a) Dándole un valor a cada posición accediendo individualmente por su índice:

```php
<?php

$familia[0]="Padre";
$familia[1]="Madre";
$familia[2]="Hijo";

echo $familia[0]; //Muestra 'Padre'
echo $familia[1]; //Muestra 'Madre'

?>
```

Otra forma de inicializar un vector, es a través del constructor array, es decir, definiendo los valores iniciales separados por comas dentro de array( ), como se muestra en los siguientse ejemplos:

```php
<?php
$pais =array("Argentina","Uruguay","Brasil","Chile");
$familia = array("Padre","Madre","Hijo");

echo $familia[0]; //Muestra 'Padre'
echo $pais[1]; //Muestra Uruguay
?>
```

Los índices de los array **siempre empiezan desde 0**, así que hay que tener en cuenta que si se quiere acceder al segundo elemento contenido en un array, habrá que acceder a él mediante el índice 1, por ejemplo.

Existen otras maneras de inicializar vectores en PHP:

```php
$Pais[] = "Argentina";
$Pais[] = "Uruguay";
$Pais[] = "Brasil";
$Pais[] = "Chile";
```

En este caso se observa que no es necesario colocar el número de índice, ya que PHP lo asigna automáticamente para cada valor, comenzando siempre desde cero.

También se puede definir un arreglo asociando explícitamente el índice a un valor, como se indica a continuación:

```
$Frutas = array(0 => "Manzana",
1 => "Naranja",
2 => "Pera",
3 => "Ananá");
```

Además, los índices, pueden no ser obligatoriamente consecutivos, ni tampoco comenzar de cero, ni tampoco ser un número. (Ver ejemplos más adelante).

Se puede conocer la cantidad de elementos que tiene un vector, para ello se utiliza la función **count(vector)**. Esta función acepta como parámetro el nombre del vector y devuelve la cantidad de elementos del mismo.

Ejemplo:

Almacenar los nombres de los días de la semana en un vector y luego imprimirlos uno debajo de otro.

```
<html>
<title> ejmeplo 1 </title>
<body>
<?php
// inicializacion del vector
$dia[0] = "domingo";
$dia[1] = "lunes";
$dia[2] = "martes";
$dia[3] = "miércoles";
$dia[4] = "jueves";
$dia[5] = "viernes";
$dia[6] = "sábado";
// impresion del vector
for($i=0; $i<7; $i++)
{
echo ($dia[$i] . "<br/>") ;
}
?>
</body>
</html>
```

Se inicializa el vector indicando el número que le corresponde a cada posición entre corchetes [ ] y asignando el valor que se desea almacenar en dicha posición.

Un vector, en PHP, puede contener elementos de distintos tipos de datos, es decir, un elemento puede ser un número entero, otro una cadena, otro un número con decimales, etc. Un modelo de este caso se puede observar en el siguiente ejemplo.

Ejemplo:

Almacenar en un vector los datos personales de un empleado y luego mostrarlos en pantalla.

```
<html>
<title> ejemplo 2 </title>
<body>
<?php
// inicializacion del vector
$empleado[0] = 4371;
$empleado[1] = "martinez leandro";
$empleado[2] = "27.643.742";
$empleado[3] = 1429.54;
$empleado[4] = "arquitecto";
// impresion del vector
echo ("legajo: " . $empleado[0] . "<br/>");
echo ("nombre: " . $empleado[1] . "<br/>");
echo ("dni : " . $empleado[2] . "<br/>");
echo ("sueldo: " . $empleado[3] . "<br/>");
echo ("profesion: " . $empleado[4] . "<br/>");
?>
</body>
</html>
```

Ejemplo:

Cargar en un vector artículos de librería y luego imprimir la cantidad de ellos.

```
<html>
<title> ejemplo 3 </title>
<body>
```

```
<?php
// inicializacion del vector
$articulos =array("lápis","goma","hoja","tinta");
// impresion del vector
$cantidad = count($articulos);
echo ("la cantidad de artículos son: " . $cantidad);
?>
</body>
</html>
```

Un vector en PHP puede tener elementos en cualquier posición, por lo tanto, se puede cargar un vector con posiciones **no** consecutivas, sino en forma totalmente aleatoria.

Para poder recorrer este tipo de vectores se utilizan las funciones **next()** y **prev()**.

Ejemplo:

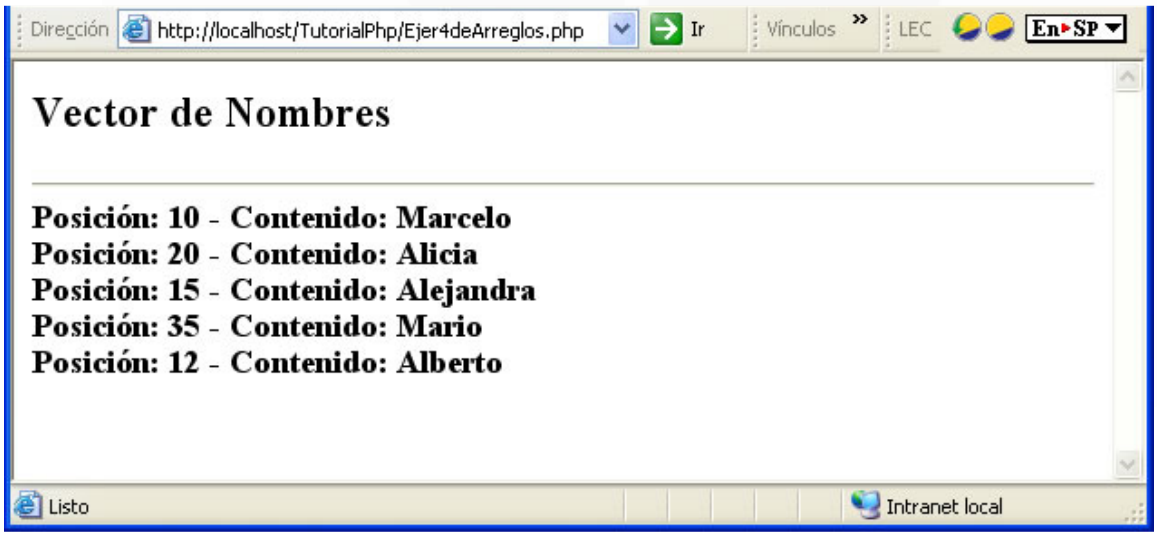
Cargar los nombres de personas en cualquier posición.

```
<html>
<title> ejemplo 4 </title>
<body>
<?php
// inicialización del vector
$nombre[10] = "marcelo";
$nombre[20] = "alicia";
$nombre[15] = "alejandra";
$nombre[35] = "mario";
$nombre[12] = "alberto";
// impresión del vector
reset($nombre);
echo ("<h2>" . "vector de nombres"."</h2>");
do
{
// buscar posición especificada
$i = key($nombre);
// buscar contenido en esa posición
```

```
$valor = current($nombre);
echo ("posición: " . $i . " - ");
echo ("contenido: " . $valor);
echo ("<br>");
}
while (next($nombre));
?>
</body>
</html>
```

Aquí se observa que los nombres fueron cargados en posiciones aleatorias. Con la función **reset(**$vector**)**, se logra llevar el puntero al principio del vector y para poder encontrar la posición de un elemento se utiliza la función **key(**$vector**)**, la cual acepta como parámetro el vector y devuelve la posición. Luego para poder obtener el contenido en esa posición se utiliza la función **current(**$vector**)**, que acepta el vector y devuelve el valor almacenado. La función **next(**$vector**)** avanza el puntero a la posición siguiente, si se ha llegado al final del vector, esta función devuelve **false**, **each(**$vector**)** nos devuelve el elemento que marca el puntero interno y mueve este puntero al siguiente elemento en el arreglo. Si ya no quedan elementos que devolver devuelve falso.

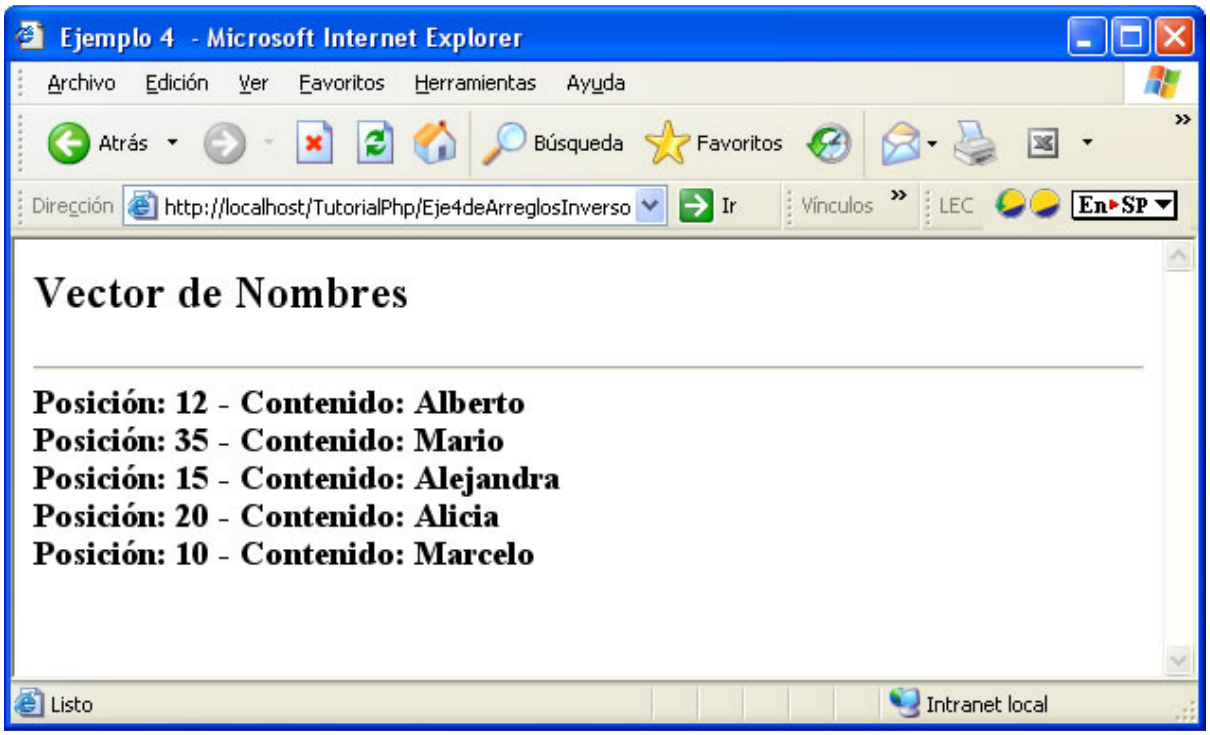
La salida de este ejemplo es la siguiente:



En caso de querer imprimir los datos del vector en forma invertida, solamente se debe cambiar la instrucción **reset($Nombre)** por: **end($Nombre)** y la instrucción: **while (next($Nombre))** por : **while (prev($Nombre))**.

Como se dijo anteriormente, las funciones **next()** y **prev()** devuelven **false** al llegar al final y principio del vector respectivamente, pero también devuelven **false** cuando el valor de un elemento es cero.

La salida quedará de la siguiente forma:



La función **end()** lleva el puntero al final del vector y con la función **prev()**, se retrocede el puntero hasta el anterior. Cuando se llegó al principio del vector, la función **prev()** devuelve **false** y sale del ciclo.

Otra forma de recorrer un vector de estas característica es utilizando las funciones **list()** y **each()**, de la siguiente manera:

Ejemplo:

Cargar Legajos de personas con sus respectivos Nombres

```
<html>
```

```
<title> ejemplo5.php </title>
<body>
<?php
// inicialización del vector
$nombre[100] = "Javier";
$nombre[200] = "Cintia";
$nombre[150] = "Ricardo";
$nombre[350] = "Raúl";
$nombre[120] = "Guillermo";
// impresión del vector
reset($nombre);
echo "<h2>"."vector de nombres"."</h2>";
while (list($i,$valor)=each($nombre))
{
echo "legajo: " . $i . " - ";
echo "nombre: " . $valor;
echo "<br/>";
}
?>
</body>
</html>
```

La función list(), almacena en los parámetros ($i, $Valor), el índice y el valor devuelto por la función each(), que tiene como parámetro el vector puesto en juego. Además, la función list(), avanza automáticamente el puntero al siguiente elemento del vector, y en caso de que el vector haya llegado al final, la función devuelve false.

Veamos otros ejemplos de utilización de estas funciones, para terminar de comprender su uso. Por ejemplo para recorrer un arreglo:

```
<?php
$ arreglo =array("lápis","goma","hoja","tinta");
reset($arreglo);
for ($i = 0; $i < count($arreglo); $i++) {
echo key($arreglo) . "=" . current($arreglo) . "<br/>";
next($arreglo);
}
?>
```

O recorrerlo en sentido inverso:

```
<?php
$ arreglo =array("lápiz","goma","hoja","tinta");
end($arreglo);
for ($i = 0; $i < count($arreglo); $i++) {
echo key($arreglo) . "=" . current($arreglo) . "<br/>";
prev($arreglo);
}
?>
```

Saltando un elemento:

```
<?php
$ arreglo =array("lápiz","goma","hoja","tinta");
reset($arreglo);
for ($i = 0; $i < count($arreglo) / 2; $i+=2) {
echo key($arreglo) . "=" . current($arreglo) . "<br/>";
next($arreglo);
next($arreglo);
}
?>
```

En un vector ordenado, estas funciones no son necesarias, pero en un vector desordenado, o por ejemplo, en un vector, donde las posiciones no vienen dadas por números, si no por nombres, como veremos a continuación, necesitamos utilizar esas funciones.

Ejemplos:

```
<?php
$v[0] = 'PHP';
$v[1] = 'estamos';
$v[2] = 'aprendiendo, paso a paso';

//Para mostrar los elementos del vector por pantalla, de forma manual:
echo $v[0].' '.$v[1].' '.$v[2];
?>
```

Pero, ¿qué pasa si no sabemos cuántos elementos hay, o hay demasiados como para ponerlos a mano? Aquí es donde entra el bucle for y el comando count.

```
<?php
$v[0] = 'PHP';
$v[1] = 'estamos';
$v[2] = 'aprendiendo, paso a paso';

$size = count($v);

for($i=0;$i<$size;$i++){
echo $v[$i].' ';
}
?>
```

Recordando el funcionamiento del bucle for, lo que hemos hecho es guardar en la variable $size el valor devuelvo por count($v), es decir, el número de elementos del vector.

Iniciamos la variable $i en cero, repetimos el bucle una vez por cada elemento que contenga el vector ($i<$size) e incrementamos el contador ($i++).

Para mostrar el contenido del vector, a cada repetición del bucle mostramos un elemento. Como la variable $i contiene el número de la repetición en curso, lo usamos para determinar el elemento del vector al que acceder ($v[$i]).

De esta forma, en la primera repetición del bucle, $i vale 0, por tanto, accedemos a la posición 0 del vector, en la segunda, $i vale 1, y accedemos a la posición 1 del vector, etc.

Pero, ¿qué ocurre si los elementos no son consecutivos o no son numéricos? Entonces debemos utilizar el resto de comandos mencionados arriba.

Ejemplos:

```
<?php
$v[0] = 'PHP';
```

```
$v[1] = 'estamos';
$v[3] = 'aprendiendo, paso a paso';

$size = count($v);
reset($v);

for($i=0;$i<$size;$i++){
$j = key($v);
echo $v[$j].' ';
next($v);
}
?>
```

En este caso, reiniciamos la posición del puntero con **reset($v)**, contamos el número de elementos que hay en el vector, y creamos un bucle con un número de repeticiones igual al número de elementos.

Guardamos en **$j** la posición actual del vector usando **key($v)**, la mostramos y avanzamos una posición el puntero del vector con **next($v)**.

También se puede hacer con un **do while:**

```
<?php
$v[0] = 'PHP';
$v[1] = 'estamos';
$v[3] = 'aprendiendo, paso a paso';
reset($v);
do{
$i = key($v);
echo $v[$i].' ';
} while(next($v));
?>
```

Esta vez lo que hemos hecho es reiniciar la posición del puntero del vector a la primera posición de este, usando **reset($v)**, luego, hemos creado un bucle **do while**.

Dentro del bucle, almacenamos la posición a mostrar del vector, en la variable **$i**, con **key($v)**. Tras mostrar el resultado, y cerrar el bucle, hemos puesto la condición para que este se repita, la cual es **next($v)**.

Creando este comando como condición, mientras queden elementos por mostrar, avanzará la posición del puntero tras cada repetición del bucle. Cuando no queden más elementos por mostrar, devolverá **false**, por lo que el bucle no se repetirá más.

Al igual que existe la función **reset** y **next**, también existe la función **end**, que mueve el puntero a la última posición del vector, y la función **prev**, que lleva el puntero a la posición anterior a la actual.

Otra forma de recorrer los vectores, es mediante los comandos **list** y **each**. Veamos su uso:

```
<?php
$lugar[0] = 'Londres';
$lugar[15] = 'Paris';
$lugar [5] = 'Valencia';
$lugar [500] = 'Bilbao';
$lugar[30] = 'Atenas';
while (list($i,$valor)=each($lugar)){
echo "$valor<br/>";
}
?>
```

Esta vez, hemos usado una única expresión: **list($i,$valor)=each($lugar)**. El comando list recibe **dos** variables, una para el contador del puntero del vector y otra que contiene el valor del vector en cada una de las posiciones de este.

Es decir, en $i se guarda la posición actual del vector, y en $valor se guarda el contenido de la posición $i del vector, que se lo hemos pasado a la función **each**, en este caso $lugar.

La función completa, de forma genérica, sería:

list($posicion_vector,$valor_vector)=each($vector)

La expresión **$valor**, dentro del bucle, se puede sustituir por **$lugar[$i]**, aunque no es más práctico hacerlo así.

Para empezar con los vectores, recomiendo utilizar y practicar el primer método mencionado en esta entrada. Es el más simple y el que menos conocimientos requiere.

Los últimos métodos, que incluyen algunas funciones, son más para cuando los vectores no están ordenados, en cuyo caso el último método parece el más sencillo.

ARRAYS ASOCIATIVOS

Hemos visto anteriormente lo que son los arrays y a como operar con ellos de forma muy elemental. Hemos aprendido que los arrays se asignan a variables.

Estas variables no tienen asignados valores, sino elementos de array que son datos que están asociados, a su vez, a un elemento del array llamado índice.

Este índice se caracteriza por conectar los elementos del array por medio de una numeración que empieza por cero. Así, el primer elemento del array tiene índice cero, el segundo tiene índice uno, y así sucesivamente.

Pero en realidad, resulta que este índice es numérico solo por defecto; es decir, tenemos la posibilidad de crear nuestro propio índice dentro de un array. Cuando hacemos esto, estamos convirtiendo el array en un array asociativo.

```
<?php
$menu = array(
"Primer plato" => "Rabas",
"Segundo plato" => "Pejerrey al verdeo con papas rústicas",
"Postre" => "Helado"
);
echo $menu["Primer plato"];
?>
```

Si repasamos el código anterior, hemos asignado a la variable `$menu` un array asociativo. Al ser asociativo tenemos que especificar un índice. El índice que hemos especificado es: *Primer plato*, *Segundo plato* y *Postre*. Posteriormente, hemos realizado una sentencia `echo` para sacar en pantalla el primer plato del menú.

En el paréntesis del array, tenemos que asociar el índice con el valor por medio del operador `=>`.

Después, para acceder a un elemento del array asociativo tenemos que escribir la variable asignada al array, y posteriormente escribir entre corchetes el índice que hemos asignado a dicho elemento del array.

Tenemos la posibilidad de poner, para especificar el índice, cualquier tipo de dato. Por defecto es un número, pero también podemos poner cadenas de texto, e incluso variables o funciones.

```
<?php
$indice = "favorito";
$color = array($indice => "violeta");
echo $color[$indice];
?>
```

Este concepto de asociar datos a elementos del array por medio del índice nos permitirá dar el siguiente paso en el tema de los arrays.

Resumiendo entonces, la diferencia de los Arrays asociativos con los indexados, es que en vez de acceder al contenido de un "compartimiento" del array por el índice de su posición, podemos definir las CLAVES con cadenas que identifiquen mejor el contenido de cada uno de esos compartimientos.

Al tener cadenas en vez de un número como clave, es mucho más fácil identificar el valor de cada elemento del array:

```
<?php
$precios["TV"]=700;
$precios["Telefono"]=150;
$precios["Computadora"]=1900;
echo $precios["Computadora"]; //Muestra '1900'
?>
```

Otra manera de definir un array asociativo es:

```
<?php
$precios=array("TV"=>1500, "Telefono"=>150, "Computadora"=> 1900);
echo $precios["Computadora"]; //Muestra '1900'
?>
```

Observación: una función muy práctica para usar con arrays es count, que devuelve la cantidad de elementos de un array.

```
<?php
$un_array=array(4,6,3,6,7,23,1);
echo count($un_array); //Muestra "7"
?>
```

Como ya sabemos, un array sencillo está formado por conjuntos de parejas índice => valor, o como suele expresarse en inglés, key, value. Hasta ahora hemos manejado un ejemplo con índices o keys numéricos (también conocidos como arrays escalares), pero también podemos usar strings como índices, es decir, cadenas de texto. Este tipo de array es el array asociativo:

```
<?php
$mis_barrios = array(
"norte"=>"Vicente Lopez",
"sur"=>"Avellaneda",
"oeste"=>"Castelar");
?>
```

Ejemplo:

Cargar en un vector algunas ciudades del mundo, de manera que el índice del vector contenga los tres primeros caracteres de la ciudad almacenada.

```
<html>
<title> ejemplo 6.php </title>
<body>
<?php
// inicialización del vector
$ciudad = array("par" => "París",
"lon" => "Londres",
"ate" => "Atenas",
"ber" => "Berlín",
"lim" => "Lima");
echo "<h2>"."Vector de ciudades"."</h2>";
while (list($i,$valor)=each($ciudad))
{
echo "posición: " . $i . " - ";
echo "contenido: " . $valor;
echo "<br/>";
}
?>
</body>
```

```
</html>
```


Ejemplo 6.php - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos
Dirección http://localhost/TutorialPhp/Eje6.Php
Ir Vínculos LEC
En SP
Vector de Ciudades
Posición: Par - Contenido: Paris
Posición: Lon - Contenido: Londres
Posición: Ate - Contenido: Atenas
Posición: Ber - Contenido: Berlin
Posición: Lim - Contenido: Lima
Listo
Intranet local

BUCLE PARA ARRAYS

Vimos la unidad pasada el concepto de bucle y los más utilizados (WHILE, FOR, DO WHILE), existe también un bucle para trabajar con arrays llamado FOREACH que pasamos a explicar.

foreach

PHP 4 introdujo un constructor *foreach*, muy parecido al de Perl y algunos otros lenguajes.

Simplemente da un modo fácil de iterar (recorrer) sobre arrays.

foreach sólo trabaja sobre arrays y dará error al intentar usarlo en una variable con un diferente tipo de datos o una variable no inicializada. Hay dos sintaxis; la segunda es una extensión menor, pero útil de la primera:

foreach ($array as $value)

sentencias

foreach ($array as $key => $value)

sentencias

La primera forma recorre el array dado ($*array*) y en cada ciclo, el valor de cada elemento del mismo es asignado a *$value* y el puntero interno del array es avanzado en uno (así en el próximo ciclo se estará mirando el siguiente elemento).

La segunda manera hace lo mismo, excepto que la clave del elemento actual será asignada a la variable *$key* en cada ciclo.

Cuando *foreach* inicia la primera ejecución, el puntero interno del array se pone automáticamente en el primer elemento del mismo. Esto significa que no es necesario llamar la función reset() antes de un bucle *foreach*.

foreach, a menos que se le indique lo contrario, opera sobre una copia del array especificado y no sobre el propio array ya que tiene algunos efectos secundarios sobre el puntero del mismo. No se puede confiar en el puntero del array durante o después del *foreach* sin reposicionarlo.

foreach no soporta la capacidad de suprimir mensajes de error usando '@'.

Algunos ejemplos más para demostrar su uso:

```php
<?php
/* foreach ejemplo 1: solo valor*/
$a = array(1, 2, 3, 17);
foreach($a as $v) {
print "Valor actual de \$a: $v.\n";
}
?>

<?php
/* foreach ejemplo 2: valor (con clave impresa para ilustrar) */
$a = array(1, 2, 3, 17);
$i = 0; /* solo para propósitos demostrativos */
foreach($a as $v) {
print "\$a[$i] => $v.\n";
$i++;
}
?>

<?php
/* foreach ejemplo 3: clave y valor */
$a = array(
"uno" => 1,
"dos" => 2,
"tres" => 3,
"diecisiete" => 17
);
foreach($a as $k => $v) {
print "\$a[$k] => $v.\n";
}
?>
```

Funciones para utilizar con Arreglos

A continuación, vamos a ver algunas funciones de utilidad para el manejo de arreglos.

Función sort()

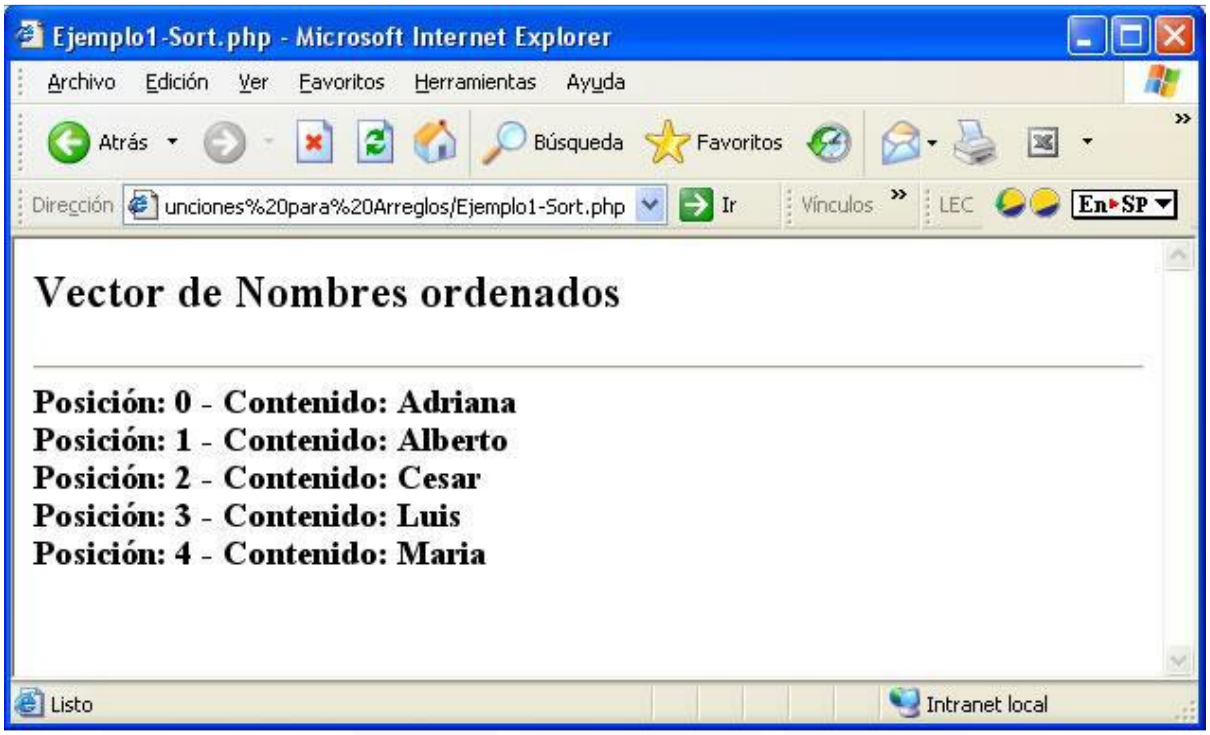
Esta función permite ordenar los elementos de un arreglo (vector) de menor a mayor, en orden numérico o alfabético, con el siguiente orden:

Primero se ordenarán los números, luego los signos de puntuación y por último las letras.

Ejemplo:

Cargar en un arreglo nombre de personas y luego mostrarlos ordenados de menor a mayor.

```
<html>
<title> ejemplo sort php </title>
<body>
<?php
// inicializacion del vector
$nombres = array("María", "Luis", "Alberto", "Cesar", "Adriana");
// ordenamiento del vector
sort($nombres);
echo "<h2>"."vector de nombres ordenados"."</h2>";
while (list($i,$valor)=each($nombres))
{
echo "posición: " . $i . " - ";
echo "contenido: " . $valor;
echo "<br/>";
}
?>
</body>
</html>
```



Nota: Si se utiliza la función sort() con índices de texto, se observa que, una vez ordenado, ha cambiado los índices de texto por números, comenzando de cero en adelante. Para salvar este inconveniente se utiliza la función asort().

Función asort()

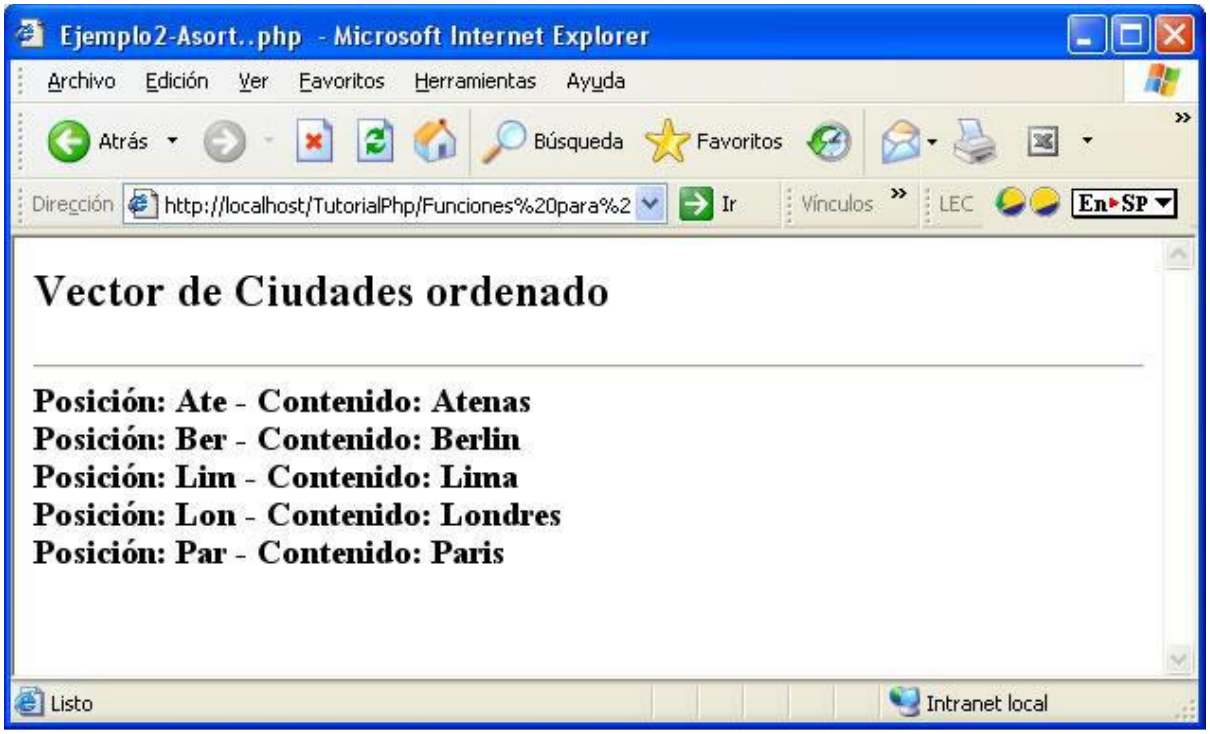
Esta función permite ordenar arreglos con índices de texto, manteniendo el valor de sus índices. Tiene como parámetro el vector, y el ordenamiento es de menor a mayor.

Ejemplo:

Ordenar alfabéticamente de menor a mayor el vector de ciudades del mundo del ejemplo Nro. 6. de la página anterior.

```
<html>
<title> ejemplo asort php </title>
<body>
```

```
<?php
// inicializacion del vector
$ciudad = array("par" => "París",
"lon" => "Londres",
"ate" => "Atenas",
"ber" => "Berlín",
"lim" => "Lima");
asort($ciudad);
echo "<h2>"."Vector de ciudades ordenado"."</h2>";
while (list($i,$valor)=each($ciudad))
{
echo "posición: " . $i . " - ";
echo "contenido: " . $valor;
echo "<br/>";
}
?>
</body>
</html>
```



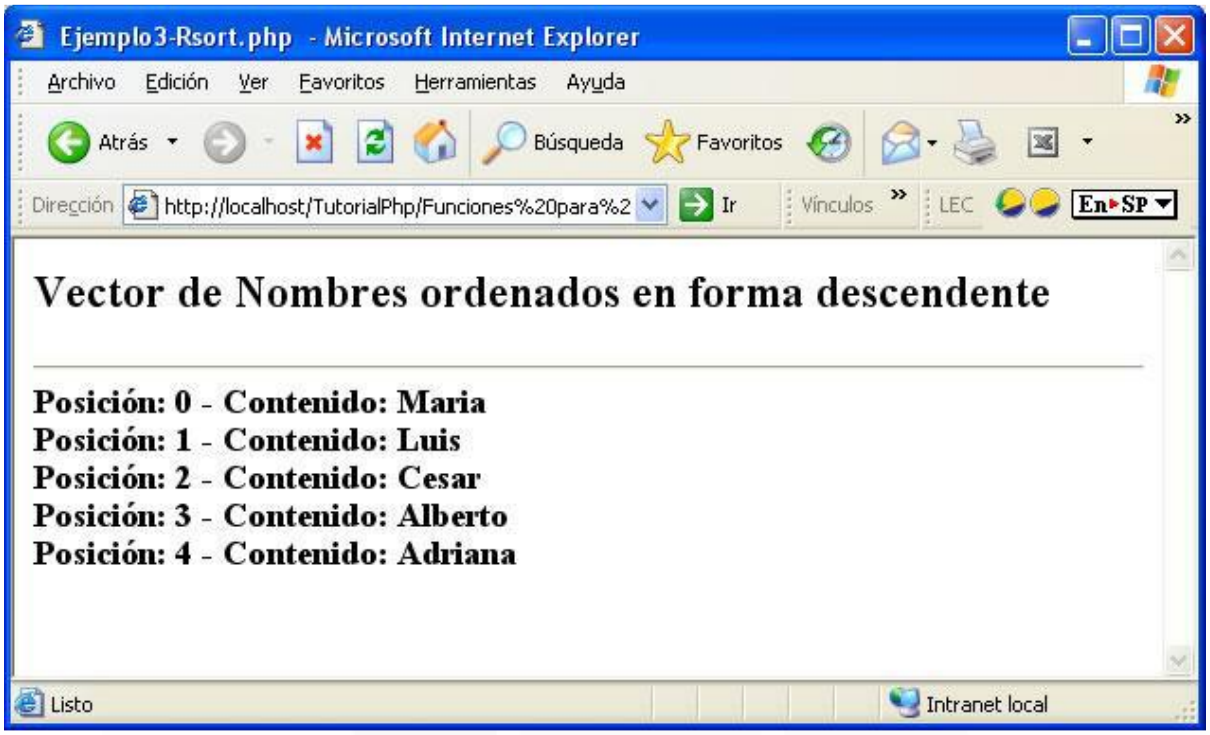
Función rsort()

Es la inversa de la función sort(). Permite ordenar un vector pasado como parámetro en forma descendente, es decir de mayor a menor.

Ejemplo:

Modificar el ejemplo anterior para mostrar el vector de nombres de personas en forma descendente.

```
<html>
<title> ejemplo3-rsort.php </title>
<body>
<?php
// inicializacion del vector
$nombres = array("María", "Luis", "Alberto", "Cesar", "Adriana");
// ordenamiento del Vector descendente
rsort($nombres);
echo "<h2>"."Vector de nombres ordenados en forma descendente "."</h2>";
while (list($i,$valor)=each($nombres))
{
echo "posición: " . $i . " - ";
echo "contenido: " . $valor;
echo "<br/>";
}
?>
</body>
</html>
```



Función arsort()

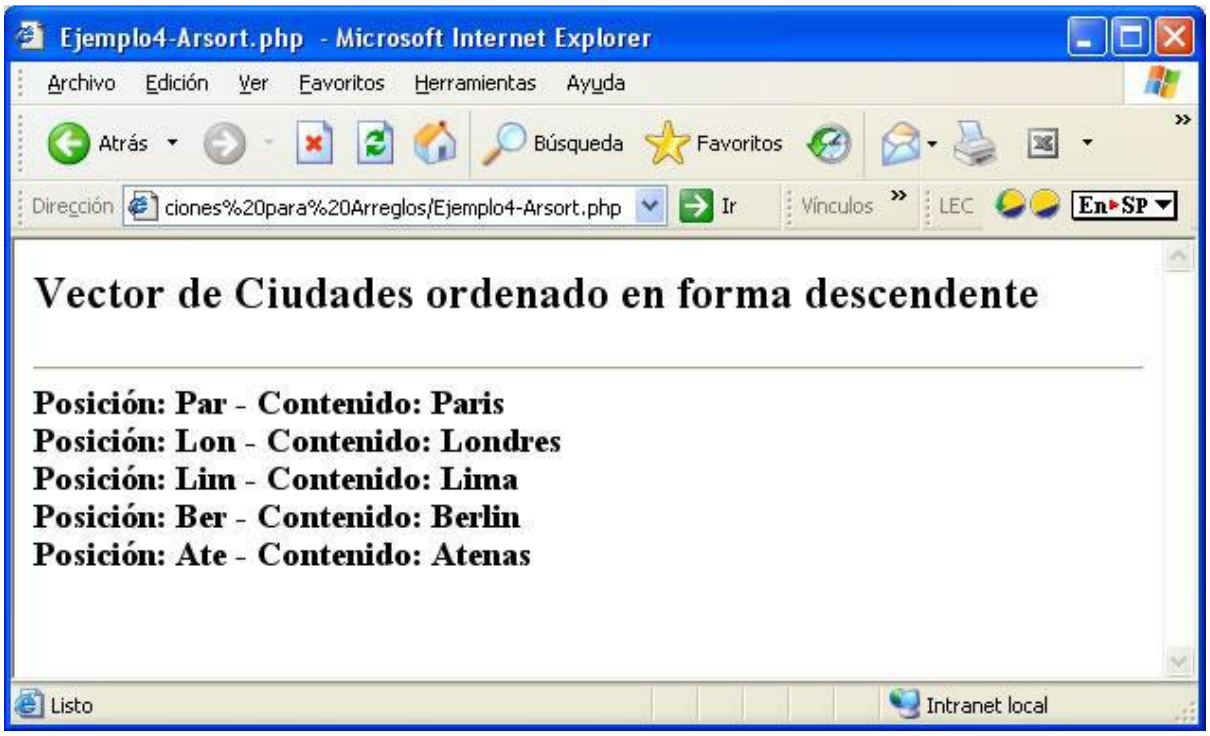
Es la inversa de la función asort(). Permite ordenar un vector con índices de texto pasado como parámetro en forma descendente, es decir de mayor a menor.

Ejemplo:

Modificar el ejemplo 2 para mostrar el vector de ciudades en forma descendente.

```
<html>
<title> ejemplo arsort php </title>
<body>
<?php
// inicializacion del vector
$ciudad = array("par" => "París",
"lon" => "Londres",
```

```
"ate" => "Atenas",
"ber" => "Berlín",
"lim" => "Lima");
arsort($ciudad);
echo "<h2>"."vector de ciudades ordenado en forma descendente"."</h2>";
while (list($i,$valor)=each($ciudad))
{
echo "posición: " . $i . " - ";
echo "contenido: " . $valor;
echo "<br/>";
}
?>
</body>
</html>
```



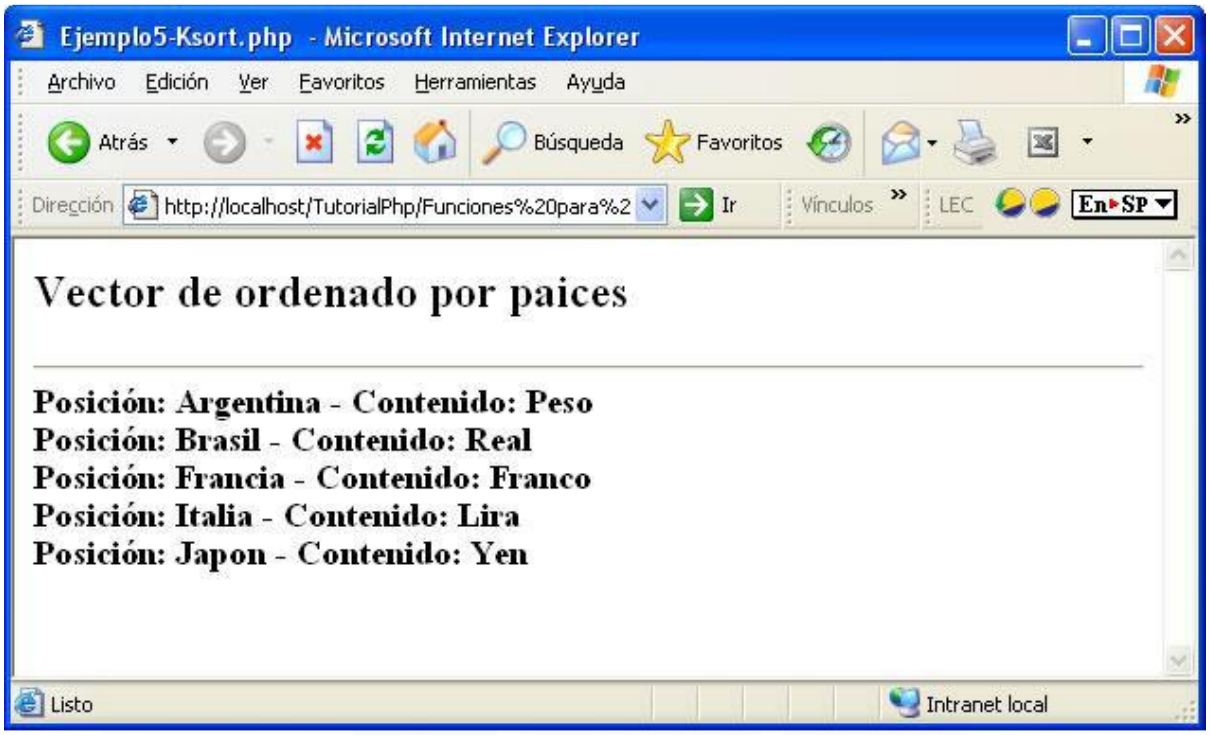
Función ksort()

Permite ordenar un arreglo por el índice del mismo, sin tener en cuenta los elementos.

Ejemplo:

Cargar nombres de países como índices con sus respectivas monedas y mostrarlos ordenados por los índices.

```
<html>
<title> ejemplo5-ksort.php </title>
<body>
<?php
// inicialización del vector
$ciudad = array("francia" => "franco",
"japon" => "yen",
"brasil" => "real",
"italia" => "lira",
"argentina" => "peso");
arsort($ciudad);
ksort($ciudad);
echo "<h2>"."Vector ordenado por países "."</h2>";
while (list($i,$valor)=each($ciudad))
{
echo "posición: " . $i . " - ";
echo "contenido: " . $valor;
echo "<br/>";
}
?>
</body>
</html>
```



Función krsort()

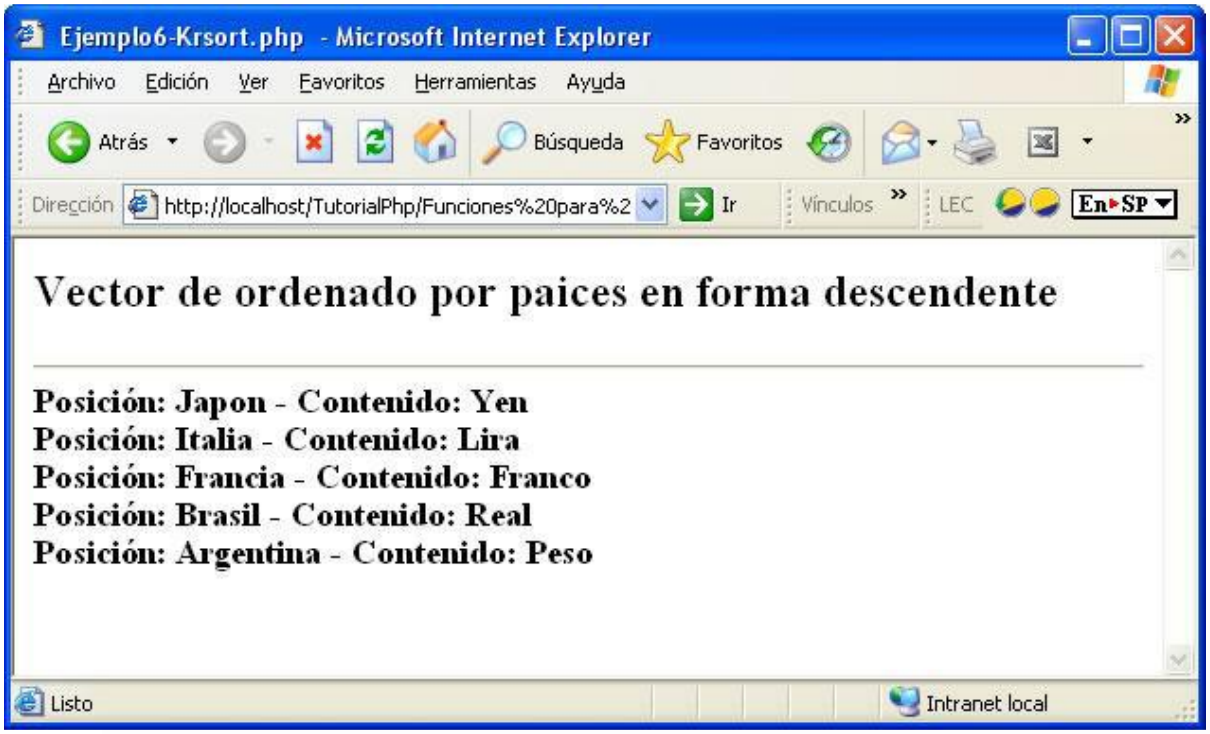
Ordena un arreglo de acuerdo al índice del mismo en orden descendente, es decir de mayor a menor.

Ejemplo:

Mostrar en orden inverso el listado del vector del ejemplo anterior.

```
<html>
<title> ejemplo krsort php </title>
<body>
<?php
// inicialización del vector
$ciudad = array("francia" => "franco",
"japon" => "yen",
"brasil" => "real",
"italia" => "lira",
"argentina" => "peso");
arsort($ciudad);
```

```
krsort($ciudad);
echo "<h2>"."vector de ordenado por países en forma descendente"."</h2>";
while (list($i,$valor)=each($ciudad))
{
echo "posición: " . $i . " - ";
echo "contenido: " . $valor;
echo "<br/>";
}
?>
</body>
</html>
```



Función shuffle()

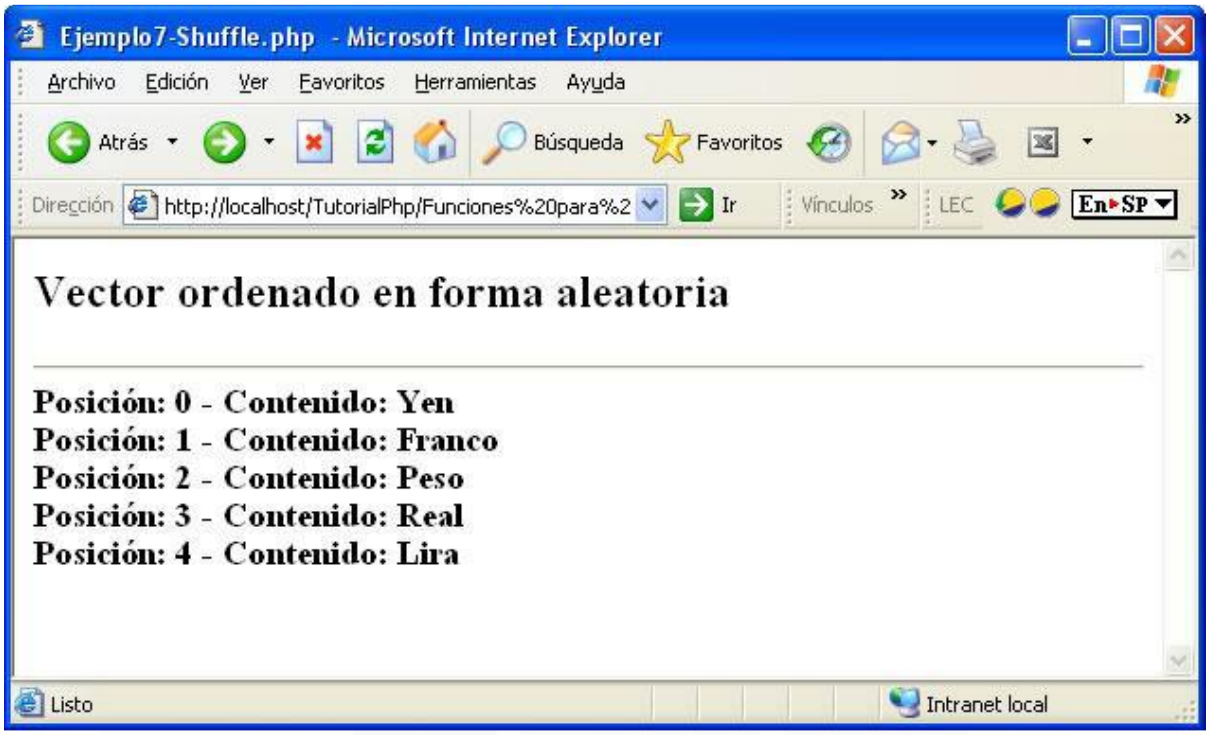
Permite mezclar los elementos de un arreglo, es decir, que cambie en forma aleatoria el orden de los elementos del arreglo que se le pase como parámetro.

Ejemplo:

Mezclar los elementos del vector del ejemplo 5.

```
<html>
<title> ejemplo shuffle php </title>
<body>
<?php

// inicialización del vector
$ciudad = array("francia" => "franco",
"japon" => "yen",
"brasil" => "real",
"italia" => "lira",
"argentina" => "peso");
arsort($ciudad);
shuffle($ciudad);
echo "<h2>"."vector ordenado en forma aleatoria "."</h2>";
while (list($i,$valor)=each($ciudad))
{
echo "posición: " . $i . " - ";
echo "contenido: " . $valor;
echo "<br/>";
}
?>
</body>
</html>
```



Nota: Se observa en este caso que las posiciones han cambiado automáticamente a valores numéricos. Comenzando por la posición cero.

Si se vuelve a ejecutar el programa, se obtiene otro orden de contenido.

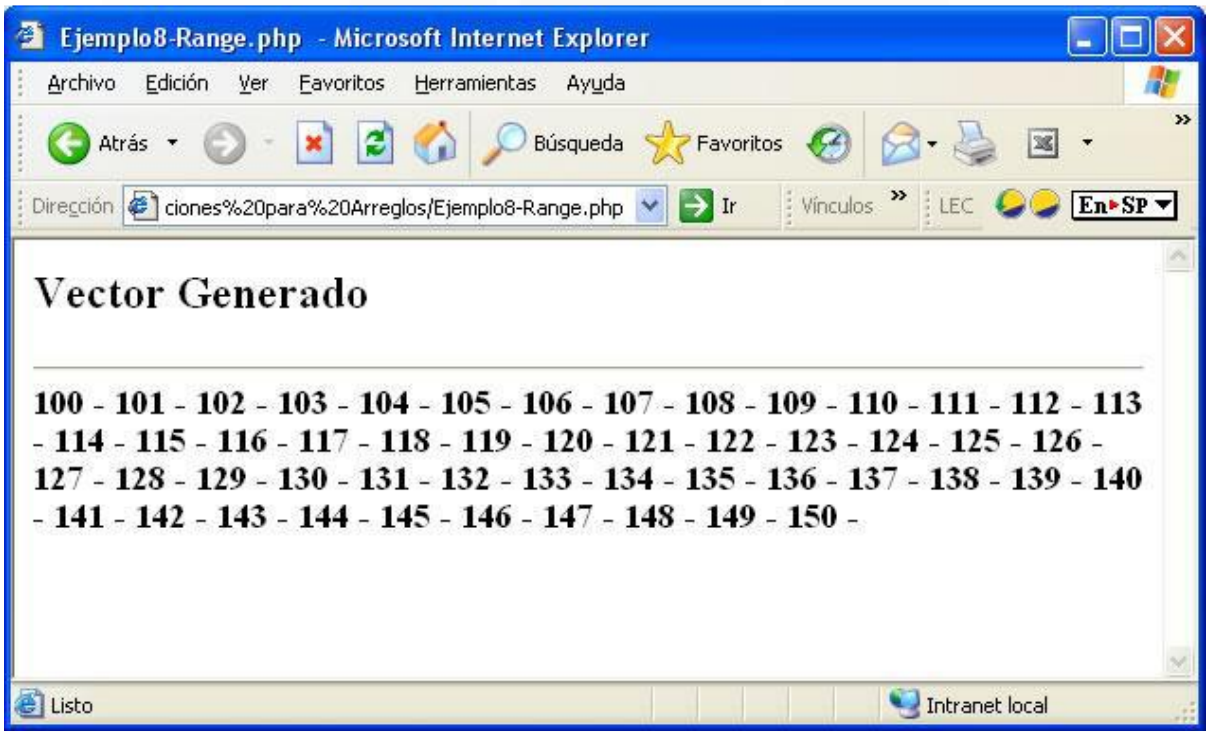
Función range()

Con esta función, se puede asignar a los elementos de un arreglo los valores comprendidos en un determinado rango. Acepta como parámetros los valores de inicio y fin de los datos a almacenar.

Ejemplo:

Generar un vector con todos los valores comprendidos entre 100 y 150. Posteriormente imprimirlos uno al lado del otro.

```
<html>
<title> ejemplo range php </title>
<body>
<?php
// inicialización del vector
$numeros = range(100,150);
echo "<h2>"."vector generado"."</h2>";
while (list($i,$valor)=each($numeros))
{
echo $valor . " - ";
}
?>
</body>
</html>
```

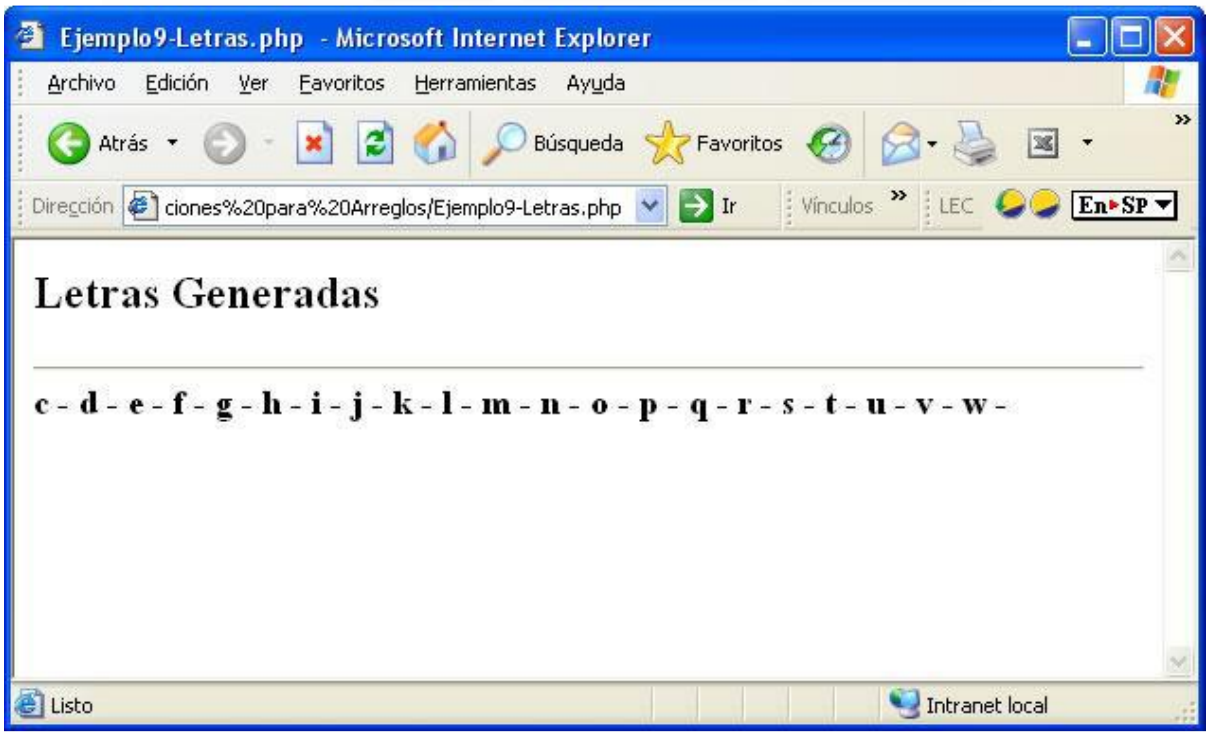


Esta función, también permite definir un rango entre letras. (Únicamente letras, si se pusiera una cadena, tomaría la primer letra).

Ejemplo:

Asignar a un vector las letras comprendidas entre “c” y “p”, y luego mostrarlas una al lado de la otra.

```
<html>
<title> ejemplo letras php </title>
<body>
<?php
// inicialización del vector
$letras = range("c","w");
echo "<h2>"."letras generadas"."</h2>";
while (list($i,$valor)=each($letras))
{
echo $valor . " - ";
}
?>
</body>
</html>
```



También se puede generar valores en orden inverso para almacenarlos en un vector, simplemente colocando en el primer parámetro de la función range(), un valor superior al segundo parámetro.

Función array_fill()

Permite almacenar en un arreglo (Vector), un dato que se va a repetir consecutivamente una cierta cantidad pasada como parámetro.

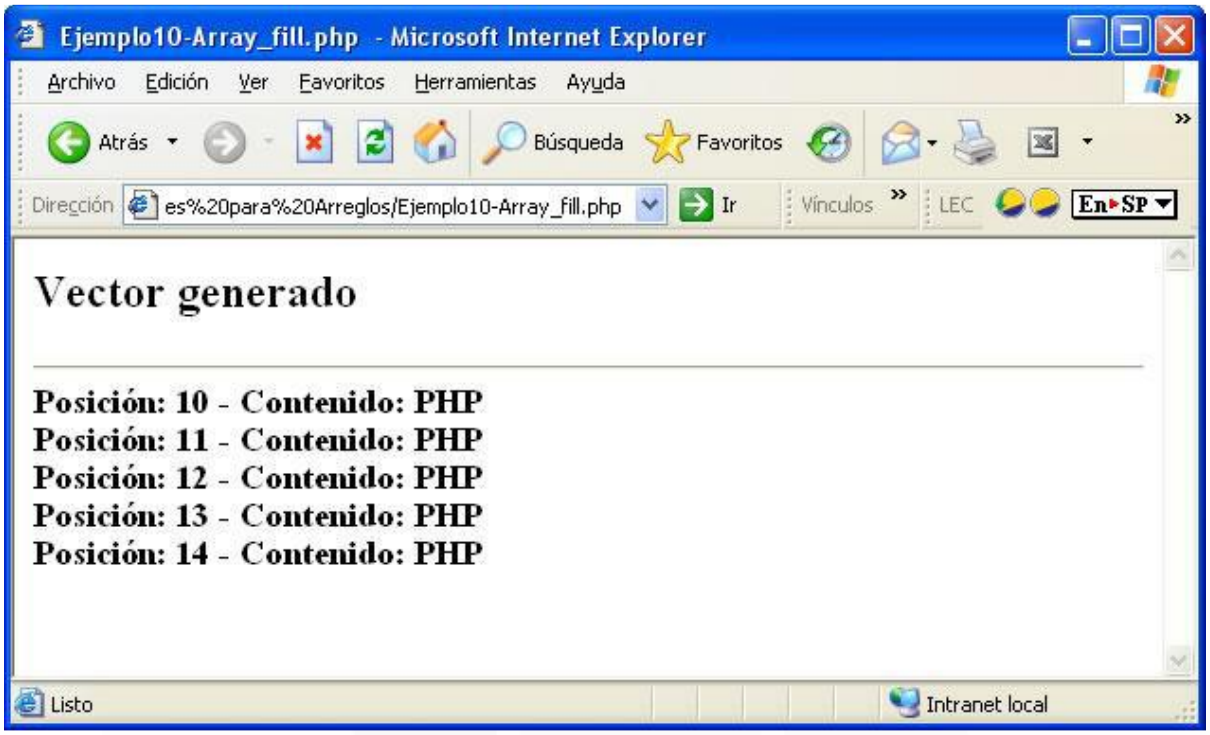
Los parámetros que acepta son tres, y tiene el siguiente formato:

$vector = array_fill (Índice, Cantidad, Dato);

Índice indica cual será el primer elemento. Cantidad indica cuantos elementos se generarán. Dato es el valor con que se llenará el vector.

Ejemplo:

```
<html>
<title> ejemplo array_fill php </title>
<body>
<?php
// inicialización del vector
$vector=array_fill(10,5,"php");
echo "<h2>"."vector generado"."</h2>";
while (list($i,$valor)=each($vector))
{
echo "posición: " . $i . " - ";
echo "contenido: " . $valor;
echo "<br/>";
}
?>
</body>
</html>
```



En este ejemplo se definió un arreglo que comience en el índice 10, que contenga 5 elementos y almacene PHP como dato.

Función in_array()

Mediante esta función, se puede buscar un dato dentro de un arreglo. Si el dato existe dentro del arreglo, la función devuelve verdadero, de lo contrario devuelve falso.

Ejemplo:

Cargar los días de la semana en un vector y luego almacenar en una variable un día cualquiera. Imprimir el vector y determinar si el contenido de la variable se encuentra incluido en el vector.

```
<html>
<title> ejemplo array php </title>
<body>
<?php
// inicialización del vector
```

```
$dias=array("domingo","lunes","martes","miércoles","jueves","viernes","sábado");
$d = "martes";
echo "<h2>"."vector de días"."</h2>";
while (list($i,$valor)=each($dias))
{
echo $valor. " ";
}
echo "<br/>";
echo "<br/>";
if (in_array($d,$dias)) {
echo "$d existe en el vector dias";
} else {
echo "$d no existe en el vector dias";
} ?>
</body>
</html>
```

MÁS FUNCIONES PARA ARRAYS

Modificar el número de elementos de un array

Ahora vamos a ver varios ejemplos mediante los cuales nuestros arrays pueden aumentar o reducir el número de casillas disponibles.

Reducir el tamaño de un array

array_slice()

Para disminuir el número de casillas de un arreglo tenemos varias funciones. Entre ellas, array_slice() la utilizamos cuando queremos recortar algunas casillas del arreglo, sabiendo los índices de las casillas que deseamos conservar.

Recibe tres parámetros. El array, el índice del primer elemento y el número de elementos a tomar, siendo este último parámetro opcional.

En el ejemplo siguiente tenemos un array con cuatro nombres propios. En la primera ejecución de array_slice() estamos indicando que deseamos tomar todos los elementos desde el índice 0 (el principio) hasta un número total de 3 elementos.

El segundo array_slice() indica que se tomen todos los elementos a partir del índice 1 (segunda casilla).

```
<?php
$entrada = array ("Miguel", "Pepe", "Juan", "Julio", "Pablo");
//modifico el tamaño
$salida = array_slice ($entrada, 0, 3);
//muestro el array
foreach ($salida as $actual)
echo $actual . "<br/>";
//modifico otra vez
$salida = array_slice ($salida, 1);
//muestro el array
foreach ($salida as $actual)
```

```
echo $actual . "<br/>";
?>
```

Tendrá como salida:

Miguel

Pepe

Juan

Pepe

Juan

array_shift()

Esta función extrae el primer elemento del array y lo devuelve. Además, acorta la longitud del array eliminando el elemento que estaba en la primera casilla. Siempre hace lo mismo, por tanto, no recibirá más que el array al que se desea eliminar la primera posición.

En el código siguiente se tiene el mismo vector con nombres propios y se ejecuta dos veces la función array_shift() eliminando un elemento en cada ocasión.

Se imprimen los valores que devuelve la función y los elementos del array resultante de eliminar la primera casilla.

```
<?php
$entrada = array ("Miguel", "Pepe", "Juan", "Julio", "Pablo");
//quito la primera casilla
$salida = array_shift ($entrada);
//muestro el array
echo "La función devuelve: " . $salida . "<br/>";
foreach ($entrada as $actual)
echo $actual . "<br/>";
//quito la primera casilla, que ahora sería la segunda del array original
$salida = array_shift ($entrada);
echo "La función devuelve: " . $salida . "<br/>";
//muestro el array
```

```
foreach ($entrada as $actual)
echo $actual . "<br/>";
?>
```

Da como resultado:

La función devuelve: Miguel

Pepe

Juan

Julio

Pablo

La función devuelve: Pepe

Juan

Julio

Pablo

unset()

Se utiliza para destruir una variable dada. En el caso de los arreglos, se puede utilizar para eliminar una casilla de un array asociativo (los que no tienen índices numéricos sino que su índice es una cadena de caracteres).

Veamos el siguiente código para conocer cómo definir un array asociativo y eliminar luego una de sus casillas.

```
<?php
$estadios_futbol = array("Barcelona"=> "Nou Camp","Real Madrid" => "Santiago
Bernabeu","Valencia" => "Mestalla", "Independiente"=> "Libertadores de América");
//mostramos los estadios
foreach ($estadios_futbol as $indice=>$actual)
echo $indice . " -- " . $actual . "<br/>";
```

```
//eliminamos el estadio asociado al real madrid
unset ($estadios_futbol["Real Madrid"]);
//mostramos los estadios otra vez
foreach ($estadios_futbol as $indice=>$actual)
echo $indice . " -- " . $actual . "<br/>";
?>
```

La salida será la siguiente:

Barcelona -- Nou Camp

Real Madrid -- Santiago Bernabeu

Valencia -- Mestalla

Independiente -- Libertadores de América

Barcelona -- Nou Camp

Valencia -- Mestalla

Independiente -- Libertadores de América

Aumentar el tamaño de un array

Tenemos también a nuestra disposición varias funciones que nos pueden ayudar a aumentar el número de casillas de un arreglo.

array_push()

Inserta al final del array una serie de casillas que se le indiquen por parámetro. Por tanto, el número de casillas del array aumentará en tantos elementos como se hayan indicado en el parámetro de la función. Devuelve el número de casillas del array resultante.

Veamos este código donde se crea un arreglo y se añaden luego tres nuevos valores.

```
<?php
```

```
$tabla = array ("Lagartija", "Araña", "Perro", "Gato", "Ratón");
//aumentamos el tamaño del array
array_push($tabla, "Gorrión", "Paloma", "Oso");
foreach ($tabla as $actual)
echo $actual . "<br/>"; ?>
```

Da como resultado esta salida:

Lagartija

Araña

Perro

Gato

Ratón

Gorrión

Paloma

Oso

array_merge()

Ahora vamos a ver cómo unir dos arrays utilizando la función array_merge(). A ésta se le pasan dos o más arrays por parámetro y devuelve un arreglo con todos los campos de los vectores pasados.

En este código de ejemplo creamos tres arrays y luego los unimos con la función array_merge().

```
<?php
$tabla = array ("Lagartija", "Araña", "Perro", "Gato", "Ratón");
$tabla2 = array ("12","34","45","52","12");
$tabla3 = array ("Sauce","Pino","Naranjo","Chopo","Perro","34");
//aumentamos el tamaño del array
$resultado = array_merge($tabla, $tabla2, $tabla3);
foreach ($resultado as $actual)
echo $actual . "<br/>";
```

```
?>
```

Da como resultado:

Lagartija

Araña

Perro

Gato

Ratón

12

34

45

52

12

Sauce

Pino

Naranjo

Chopo

Perro

34

Una última cosa.

También pueden introducirse nuevas casillas en un arreglo por los métodos habituales de asignar las nuevas posiciones en el array a las casillas que necesitemos.

En arrays indexados se haría así:

```
$tabla = array ("Sauce","Pino","Naranjo");
$tabla[3]="Algarrobo";
```

En arrays asociativos:

```
$estadios_futbol = array("Valencia" => "Mestalla","Independiente"=> "Libertadores de América");
$estadios_futbol["Barcelona"]= "Nou Camp";
```

Resumen

En esta Unidad…

En la presente unidad trabajamos con vectores.

En la próxima Unidad…

En la próxima unidad trabajaremos con funciones en PHP.