PHP-TUT-08 Matrices en PHP (II)

En un artículo anterior vimos los fundamentos de trabajo con matrices en PHP. En este artículo vamos a continuar con el estudio de este tema dado que, por las prestaciones que este lenguaje nos ofrece al efecto, las matrices son una forma estupenda de solucionar una gran cantidad de problemas. En muchísimas ocasiones me he enfrentado a problemas que se han resuelto con muy pocas líneas de código y que, en otros lenguajes, habrían supuesto ingentes horas de desarrollo.

AGREGAR O QUITAR ELEMENTOS DE UNA MATRIZ

PHP cuenta con cuatro funciones para agregar o eliminar elementos de una matriz. Funcionan tanto con matrices indexadas como asociativas, de una dimensión o de más. Son las siguientes:

La función array_pop(). Extrae el último elemento de una matriz, eliminándolo de la misma. Cómo argumento recibe la matriz sobre la que vamos a operar. Observa el código pop.php:

```
<?php
 echo "<pre>";
 $personas = array(
  array("nombre"=>"Paco", "apellido"=>"García"),
  array("nombre"=>"Andrés", "apellido"=>"Fuentes"),
  array("nombre"=>"Carmen", "apellido"=>"Torres"),
  array("nombre"=>"Susana", "apellido"=>"Gómez")
 );
 echo "La matriz inicialmente contiene:<br />";
 var_dump($personas);
 $ultimaPersona = array_pop($personas);
 echo "<br />=========================================<br /><br />";
 echo "Los datos de la última persona son:<br />";
 var_dump($ultimaPersona);
 echo "<br />=========================================<br /><br />";
 echo "Ahora la matriz contiene:<br />";
 var_dump($personas);
 echo "<br />=========================================<br /><br />";
?>
```

Observa el resultado en tu navegador. Verás cómo, en el primer volcado de la matriz en pantalla, esta sale completa, con los datos de las cuatro personas que hemos incluido. Despues de usar array_pop() tenermos el último elemento en otra variable. Cómo la matriz original es bidimensional, el último elemento es, a su vez, una matriz. Al hacer el segundo volcado de la matriz original vemos que ya solo tiene tres elementos. El cuarto ha desaparecido, tal cómo esperábamos.

La función array_shift(). Actúa del mismo modo que la anterior, con la diferencia de que opera sobre el primer elemento de la matriz, no sobre el último. Observa el listado de shift.php:

```
<?php
 echo "<pre>";
 $personas = array(
  array("nombre"=>"Paco", "apellido"=>"García"),
  array("nombre"=>"Andrés", "apellido"=>"Fuentes"),
  array("nombre"=>"Carmen", "apellido"=>"Torres"),
  array("nombre"=>"Susana", "apellido"=>"Gómez")
 );
 echo "La matriz inicialmente contiene:<br />";
 var_dump($personas);
 $primeraPersona = array_shift($personas);
 echo "<br />=========================================<br /><br />";
 echo "Los datos de la primera persona son:<br />";
 var_dump($primeraPersona);
```

```
 echo "<br />==========================================<br /><br />";
 echo "Ahora la matriz contiene:<br />";
 var_dump($personas);
 echo "<br />==========================================<br /><br />";
?>
```

Una vez más, te invito a copiar este código y ejecutarlo en tu navegador. Verás cómo esta vez hemos suprimido el primer elemento de la matriz, extrayéndolo a una variable aparte.

La función array_push(). Esta es la opuesta de array_pop(). Se emplea para agregar un elemento a una matriz, al final de la misma. Observa el listado push.php:

```
<?php
 echo "<pre>";
 $personas = array(
  array("nombre"=>"Paco", "apellido"=>"García"),
  array("nombre"=>"Andrés", "apellido"=>"Fuentes"),
  array("nombre"=>"Carmen", "apellido"=>"Torres"),
  array("nombre"=>"Susana", "apellido"=>"Gómez")
 );
 echo "La matriz inicialmente contiene:<br />";
 var_dump($personas);
 $personaNuevaAlFinal = array("nombre"=>"Lorena", "apellido"=>"Gutiérrez");
 $total = array_push($personas, $personaNuevaAlFinal);
 echo "<br />==========================================<br /><br />";
 echo "Ahora la matriz contiene ".$total." elementos:<br />";
 var_dump($personas);
 echo "<br />==========================================<br /><br />";
?>
```

Cómo ves en el código, en la línea resaltada, esta función recibe dos argumentos: el primero es la matriz sobre la que operamos, y el segundo es el elemento que vamos a agregarle al final. La función nos devuleve un valor numérico que indica el número de elementos que le quedan a la matriz después de la operación.

Carga el código en tu navegador para ver el resultado.

La función array_unshift(). Es la función opuesta a array_shift(). Agrega un elemento al principio de la matriz y, cómo la función anterior, devuelve el número de elementos que le quedan a la matriz después de la operación. Los argumentos que recibe, cómo en el caso anterior, son dos: la matriz a la que le vamos a agregar el elemento, y el propio elemento. Observa el listado unshift.php:

```
<?php
 echo "<pre>";
 $personas = array(
  array("nombre"=>"Paco", "apellido"=>"García"),
  array("nombre"=>"Andrés", "apellido"=>"Fuentes"),
  array("nombre"=>"Carmen", "apellido"=>"Torres"),
  array("nombre"=>"Susana", "apellido"=>"Gómez")
 );
 echo "La matriz inicialmente contiene:<br />";
 var_dump($personas);
 $personaNuevaAlPrincipio = array("nombre"=>"Lorena", "apellido"=>"Gutiérrez");
 $total = array_unshift($personas, $personaNuevaAlPrincipio);
 echo "<br />==========================================<br /><br />";
 echo "Ahora la matriz contiene ".$total." elementos:<br />";
 var_dump($personas);
 echo "<br />==========================================<br /><br />";
?>
```

Ejecútalo en tu navegador para ver cómo opera.
Desafortunadamente, PHP solo incluye funciones para insertar un elemento al principio o al final de una matriz, no "en medio", en una posición específica, delante o detrás de un elemento concreto. Nosotros hemos resuelto eso con un pequeño truco, tal como se detalla en el artículo de este enlace. Esperamos que te guste.

RECORRER UNA MATRIZ

Las matrices en PHP cuentan con un puntero interno que permite dessplazarse de un elemento a otro de modo secuencial. Este puntero permite recorrer la matriz, volver al elemento anterior, pasar al siguiente, etc.
Podemos usar la funcion each() para obtener cada elemento de una matriz. Observa el script each.php:

```
<?php
 echo "<pre>";

 /* Creamos una matriz inicial. */
 $personas = array(
  array("nombre"=>"Paco", "apellido"=>"García"),
  array("nombre"=>"Andrés", "apellido"=>"Fuentes"),
  array("nombre"=>"Carmen", "apellido"=>"Torres"),
  array("nombre"=>"Susana", "apellido"=>"Gómez")
 );

 /* Usamos each() para obtener cada elemento. */
 while ($persona = each($personas)) var_dump($persona);
?>
```

```
array(4) {
  [1]=>
  array(2) {
    ["nombre"]=>
    string(4) "Paco"
    ["apellido"]=>
    string(7) "García"
  }
  ["value"]=>
  array(2) {
    ["nombre"]=>
    string(4) "Paco"
    ["apellido"]=>
    string(7) "García"
  }
  [0]=>
  int(0)
  ["key"]=>
  int(0)
}
```

Este script obtiene cada elemento de la matriz $personas (sin eliminarlo de la matriz original) y lo aloja en la matriz $persona. Cuando obtiene el siguiente, este reemplaza al anterior en la matriz de destino. Hemos hecho un volcado en pantalla de cada elemento, tal cómo nos lo devuelve la función each(), para que veas cómo actúa. A la izquierda reproducimos el primer elemento, tal cómo tú lo verás en tu navegador al ejecutar este script:
Observa que lo que ha hecho each() es obtener un elemento de la matriz que le hemos pasado cómo argumento, y se ha montado su propia matriz, con el valor del elemento (accesible por las claves 1 y "value") y la clave del elemento (accesible por las claves 0 y "key").
Cuando usamos each() el puntero interno de la matriz se desplaza al siguiente elemento. Este desplazamiento se produce de modo automático. Existen funciones que nos permiten controlar el movimiento del puntero, de modo que podamos posicionarlo dónde lo necesitemos.

La función reset(). Rebobina el puntero al primer elemento de la matriz.
La función prev(). Mueve el puntero al elemento anterior al que está apuntando actualmente. Si el elemento actual es el primero de la matriz, no tiene efecto alguno.
La función next(). Mueve el puntero al elemento siguiente al que está apuntando actualmente. Si el elemento actual es el último de la

matriz, no tiene efecto alguno.
La función end(). Mueve el puntero al último elemento de la matriz.
La función **current()**]. Devuelve el elemento al que está apuntando el puntero. El efecto es el mismo que each(), con la única diferencia de que current() no desplaza el puntero.

TROCEADO, UNIDO, COMBINADO
En ocasiones necesitamos trabajar con partes de una matriz, o usar varias matrices cómo partes de un todo más amplio. Veamos las funciones que nos pueden ayudar en eso.
Si queremos fragmentar una matriz en otras más pequeñas (lo que resulta imprescindible cuando se manejan grandes cantidades de datos), la función array_chunk() es ideal para ello. Divide una matriz en fragmentos de n elementos, y crea una matriz bidimensional en la que cada elemento es uno de los fragmentos iniciales. Lo vamos a ver más claro enseguida con un ejemplo pero, antes, hablemos de los argumentos que recibe:

El primero es el nombre de la matriz original que vamos a dividir.
El segundo es el tamaño de los fragmentos, expresado en número de elementos. Cabe destacar que el último fragmento podría tener menos elementos. Por ejemplo, si vamos a dividir una matriz de 47 elementos en fragmentos de 10, el último sólo tendrá 7.
El tercer argumento, opcional, es un valor booleano. Si es true, se preservarán las claves iniciales de los elementos. Si es false (o si no ponemos nada) las claves originales se pierden.

Observa el código fragmentacion.php:

```
<?php
 $matrizCiudades = array(
  "ma"=>"Madrid",
  "to"=>"Toledo",
  "bi"=>"Bilbao",
  "co"=>"Coruña",
  "te"=>"Teruel",
  "se"=>"Sevilla",
  "ml"=>"Mallorca",
  "mg"=>"Málaga",
  "ba"=>"Badajoz",
  "bc"=>"Barcelona",
  "al"=>"Alicante",
  "cu"=>"Cuenca",
  "cc"=>"Cáceres",
  "cd"=>"Cádiz",
  "ct"=>"Ceuta",
  "tg"=>"Tarragona",
  "ov"=>"Oviedo",
  "cr"=>"Ciduad Real",
  "po"=>"Pontevedra"
 );
 $fragmentada = array_chunk($matrizCiudades, 5, true);
 echo "<pre>";
 var_dump($fragmentada);
?>
```

Cómo ves cuando ejecutes este código, la matriz de ciudades ha sido fragmentada en una matriz bidimensional, en la que cada elemento contiene cinco ciudades (porque así lo hemos indicado en el segundo argumento) menos el último, que sólo tiene cuatro. Además, se conservan las claves originales, porque así lo indica el tercer argumento.
La función array_combine() combina dos matrices que recibe como argumentos. De la primera se usan sus elementos cómo claves de

la matriz resultante y de la segunda se usan sus elementos como valores. Observa el script combinar.php:

```
<?php
 $claves = array("ma", "to", "bi", "co", "te", "se", "ml", "mg", "ba", "bc", "al", "cu", "cc", "cd", "ct", "tg", "ov", "cr", "po");
 $valores = array("Madrid", "Toledo", "Bilbao", "Coruña", "Teruel", "Sevilla", "Mallorca", "Málaga", "Badajoz", "Barcelona",
"Alicante", "Cuenca", "Cáceres", "Cádiz", "Ceuta", "Tarragona", "Oviedo", "Ciduad Real", "Pontevedra");
 $matrizCiudades = array_combine($claves, $valores);
 echo "<pre>";
 var_dump($matrizCiudades);
?>
```

Cómo ves, tenemos la matriz de ciudades en dos matrices: una con las claves de los elementos y otra con sus valores. Al usar array_combine() se forma la matriz completa, que puedes ver en tu navegador.

ATENCIÓN. Es imprescindible que ambas matrices, la de las claves y la de los valores, tengan el mismo número de elementos, o se producirá un error.

En gran número de ocasiones es necesario determinar que elementos son comunes a dos o más matrices. Para ello contamos con la función array_intersect(). Cómo argumentos recibe el nombre de dos o más matrices, separados por comas. Nos devuelve una matriz con los elementos de la primera que se hallan presentes en todas las demás. Observa el script comunes.php:

```
<?php
 $matriz_1 = array("rojo", "verde", "azul");
 $matriz_2 = array("naranja", "verde", "azul", "amarillo");
 $matriz_3 = array("blanco", "verde", "rosado", "fucsia");
 $resultado = array_intersect($matriz_1, $matriz_2, $matriz_3);
 echo "<pre>";
 var_dump ($resultado);
?>
```

Cuando ejecutes este código en tu navegador verás que te devuelve una matriz con un solo elemento, el único que es común a las tres matrices que hemos usado en la función.

En ocasiones tenemos resultados que son partes de un todo. Por ejemplo, podemos tener los nombres de una lista de personas en varias matrices, digamos que provenientes de diferentes consultas hechas a una misma base de datos (esta es una situación muy común en aplicaciones de cierta complejidad). Al final, tras haber efectuado las consultas y obtenido las matrices, debemos unirlas en una sóla, para procesarla. Para ello empleamos la función array_merge(), que recibe, cómo argumentos, los nombres de las matrices que vamos a combinar, devolviendo la matriz con todos los datos. Observa el script mergeado.php.

```
<?php
 $matriz_1 = array("rojo", "verde", "azul");
 $matriz_2 = array("naranja", "verde", "azul", "amarillo");
 $matriz_3 = array("blanco", "verde", "rosado", "fucsia");
 $resultado = array_merge($matriz_1, $matriz_2, $matriz_3);
 echo "<pre>";
 var_dump ($resultado);
?>
```

Observa el resultado en tu navegador para que compruebes cómo se han unido las tres matrices originales en una con todos los elementos.

También es muy fácil (y necesario con mucha frecuencia) determinar que elementos de una matriz no se encuentran en otras matrices. Para ello contamos con la función array_diff(), que recibe, cómo argumentos, los nombres de, al menos, dos matrices.

Tomando como base la primera de ellas, nos devolverá una matriz con los elementos que no se encuentran en ninguna de las demás.
Observa el script diferencia.php:

```
<?php
 $matriz_1 = array("rojo", "verde", "azul");
 $matriz_2 = array("naranja", "verde", "azul", "amarillo");
 $matriz_3 = array("blanco", "verde", "rosado", "fucsia");
 $resultado = array_diff($matriz_1, $matriz_2, $matriz_3);
 echo "<pre>";
 var_dump ($resultado);
?>
```

Pruébalo en tu navegador para ver como se comporta.

DE MATRIZ A CADENA Y VICEVERSA

En ocasiones tenemos una matriz de datos y necesitamos convertirla a una cadena de texto que contenga los valores de los elementos separados por comas (u otro separador). Esto se hace cuando necesitamos, por ejemplo, una cadena con los valores para consultas contra bases de datos o similar. La forma de hacerlo es con la función implode(). Cómo argumento recibe el nombre de una matriz, y devuelve una cadena con los valores de los elementos, todos seguidos, uno a continuación de otro, sin separación.Si los queremos separados comas u otros caracteres o secuencias, debemos añadir, ANTES(1) del nombre de la matriz, un argumento con el separador. Observa el script encadenados.php:

(1) En realidad, esta función acepta, por razones de retrocompatibilidad, los argumentos en cualquier orden, pero eso puede cambiar en el futuro, por lo que es mejor acostumbrarse a usar el orden que indicamos aquí.

```
<?php
 $matrizColores = array("rojo", "verde", "azul", "amarillo", "rosado", "fucsia");
 $cadenaColores_1 = implode($matrizColores);
 $cadenaColores_2 = implode(",", $matrizColores);
 $cadenaColores_3 = implode(" *-* ", $matrizColores);
 echo "<pre>";
 var_dump ($cadenaColores_1, $cadenaColores_2, $cadenaColores_3);
?>
```

Una vez más, te invito a que compruebes el resultado en tu navegador, y cotejes lo que obtienes con el código del script.
La función contraria es explode(). A partir de una cadena de elementos, separados por comas u otro separador, genera una matriz.
Observa el script de_cadena_a_matriz.php:

```
<?php
 $cadenaDeColores = "rojo,verde,azul,amarillo,rosado,fucsia";
 $matrizDeColores = explode(",", $cadenaDeColores);
 echo "<pre>";
 var_dump ($matrizDeColores);
?>
```

Esta función recibe dos argumentos. El primero es el separador, aquellos caracteres de la cadena que delimitarán los elementos. El segundo argumento es la propia cadena. Cómo ves en tu navegador, se genera una matriz con los elementos que estaban en la cadena.
Además,, esta función puede recibir un tercer argumento, opcional, numérico. Sería el máximo de elementos que tendrá la matriz resultante. Si la cadena tiene más elementos, los que "sobren" se agruparán en el último elemento de la matriz.

CONSIDERACIONES FINALES

Las matrices tiene otras funciones más secundarias (menos utilizadas). En sucesivos artículos las iremos comentando, cuando nos sean necesarias o útiles. En este artículo hemos incluido las más empleadas en el día a día, y no queremos alargarlo innecesariamente. Nos vemos en el próximo artículo.