Índice

Funciones de matrices/arrays

En este documento se comentan varias funciones útiles para trabajar con matrices o arrays.

Comparar arrays

Para comparar arrays existen varias funciones de la familia arraydiff.

array_diff

array_diff (array $array1, array $array2 [, array $...]): array
Compara **$array1** con uno o más arrays y devuelve los valores de **$array1** que no estén presentes en cualquiera de los demás arrays.

```
<?php
$animales = array (
    "gato" => "Garfield",
    "perro" => "Laika",
    "mono" => "Cobby",
    "Mickey",
    "oso" => "Yogui"
);
$animales2 = array (
    "perro" => "Laika",
    "gato" => "Garfield",
    "conejo" => "Yogui",
    "Donald"
);

$diferencia = array_diff($animales, $animales2);
var_dump($diferencia);
?>
```

```
Array(2) {
  ["mono"]=>
  string(5) "Cobby"
  [0]=>
  string(6) "Mickey"
}
```

Se observa que lo que compara son los valores.

array_diff_assoc

array_diff_assoc (array $array1, array $array2 [, array $...]): array
Compara **$array1** con **$array2** y devuelve la diferencia, pero esta vez comparando valores y claves.

```php
<?php
$animales = array (
    "gato" => "Garfield",
    "perro" => "Laika",
    "mono" => "Cobby",
    "Mickey",
    "Donald",
    "oso" => "Yogui"
);
$animales2 = array (
    "perro" => "Laika",
    "gato" => "Garfield",
    "conejo" => "Yogui",
    "Donald"
);

$diferencia = array_diff_assoc($animales, $animales2);
var_dump($diferencia);
?>
```

```
Array(4) {
  ["mono"]=>
  string(5) "Cobby"
  [0]=>
  string(6) "Mickey"
  [1]=>
  string(6) "Donald"
  ["oso"]=>
  string(5) "Yogui"
}
```

"Donald" también lo añade porque tienen diferente clave. En el primer array tiene clave 1 y en el segundo clave 0.

array_diff_key

array_diff_key (array $array1, array $array2 [, array $...]): array
Compara las claves de **$array1** con las claves de **$array2** y devuelve la diferencia. Es como la función array_diff(), pero sólo comparando las claves en lugar de los valores.

```php
<?php
$animales = array (
    "gato" => "Garfield",
    "perro" => "Laika",
    "mono" => "Cobby",
```

```
    "Mickey",
    "Donald",
    "oso" => "Yogui"
);
$animales2 = array (
    "perro" => "Laika",
    "gato" => "Garfield",
    "conejo" => "Yogui",
    "Donald"
);

$diferencia = array_diff_key($animales, $animales2);
var_dump($diferencia);
?>
```

```
Array(3) {
  ["mono"]=>
  string(5) "Cobby"
  [1]=>
  string(6) "Donald"
  ["oso"]=>
  string(5) "Yogui"
}
```

Unir arrays

Para unir dos arrays en PHP se pueden emplear diferentes métodos.

Operador suma

Con el operador de suma +, que ya se ha comentado previamente, se eliminan los elementos duplicados que aparecen en el array a la derecha del operador.

```
<?php
$animales = array (
    "gato" => "Garfield",
    "perro" => "Laika",
    "Mickey",
    1 => "Donald"
);
$animales2 = array (
    "gato" => "Lulú",
    "cocodrilo" => "Juancho",
    "conejo" => "Roger Rabbit",
    "Pluto"
);
```

```
$union = $animales + $animales2;
var_dump($union);
?>
```

```
Array(6) {
  ["gato"]=>
  string(8) "Garfield"
  ["perro"]=>
  string(5) "Laika"
  [0]=>
  string(6) "Mickey"
  [1]=>
  string(6) "Donald"
  ["cocodrilo"]=>
  string(7) "Juancho"
  ["conejo"]=>
  string(12) "Roger Rabbit"
}
```

Si la clave ya está cogida no lo incluye en **$union**, independientemente del valor.

array_merge

array_merge (array $array1 [, array $...]): array
Combina dos o más arrays, de modo que los valores de uno se unen al final del anterior. Devuelve el array resultante. Al contrario que con el operador suma, si los arrays de entrada tienen las mismas claves de tipo string, el último valor para esa clave sobreescribirá al anterior. Pero si son arrays numéricas, el último valor no sobreescribirá al valor original, sino que se añadirá al final.

```
<?php
$animales = array (
    "gato" => "Garfield",
    "perro" => "Laika",
    "Mickey",
    1 => "Donald"
);
$animales2 = array (
    "gato" => "Lulú",
    "cocodrilo" => "Juancho",
    "conejo" => "Roger Rabbit",
    "Pluto"
);

$union = array_merge($animales, $animales2);
```

```
var_dump($union);
?>
```

```
Array(7) {
  ["gato"]=>
  string(5) "Lulú"
  ["perro"]=>
  string(5) "Laika"
  [0]=>
  string(6) "Mickey"
  [1]=>
  string(6) "Donald"
  ["cocodrilo"]=>
  string(7) "Juancho"
  ["conejo"]=>
  string(12) "Roger Rabbit"
  [2]=>
  string(5) "Pluto"
}
```

Ejemplo de **array_merge()** con dos arrays numéricos.

```
<?php
  $uno = [1, 2, 4];
  $dos = [1, 3, 4, 9];

  $final = array_merge($uno, $dos);

  echo "<pre>";
  print_r($final);
  echo "</pre>";
?>
```

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 4
    [3] => 1
    [4] => 3
    [5] => 4
    [6] => 9
)
```

array_merge_recursive

array_merge_recursive (array $array1 [, array $...]): array
Es igual que **array_merge()**, salvo que cuando los arrays de entrada tienen las mismas claves de tipo string, los valores de estas claves se unen en un array de forma recursiva, de forma que si uno de los valores es un array, la función unirá también ésta con la correspondiente entrada de otro array. Si los arrays tienen la misma clave numérica, el valor posterior no sobreescribirá el valor original, sino que se añadirá al final.

```
<?php
$animales = array (
    "gato" => "Garfield",
    "perro" => "Laika",
    "Mickey",
    1 => "Donald"
);
$animales2 = array (
    "gato" => "Lulú",
    "cocodrilo" => "Juancho",
    "conejo" => "Roger Rabbit",
    "Pluto"
);

$union = array_merge_recursive($animales, $animales2);
var_dump($union);
?>
```

```
Array(7) {
   ["gato"]=>
  array(2) {
    [0]=>
    string(8) "Garfield"
    [1]=>
    string(5) "Lulú"
  }
  ["perro"]=>
  string(5) "Laika"
  [0]=>
  string(6) "Mickey"
  [1]=>
  string(6) "Donald"
  ["cocodrilo"]=>
  string(7) "Juancho"
  ["conejo"]=>
  string(12) "Roger Rabbit"
  [2]=>
  string(5) "Pluto"
```

```
}
```

array_combine

array_combine (array $keys, array $values): array
Crea un array utilizando un array para sus claves y otro array para sus valores. Coge los valores para crear el nuevo array (clave => valor) de los arrays de origen.

```
<?php
  $animales = array('perro', 'gato', 'leon');
  $nombres = array('Laika', 'Garfield', 'Rodolfo');

  $array = array_combine($animales, $nombres);
  var_dump($array);
?>
```

```
Array(3) {
   ["perro"]=>
  string(5) "Laika"
  ["gato"]=>
  string(8) "Garfield"
  ["leon"]=>
  string(7) "Rodolfo"
}
```

Rellenar arrays

Las siguientes funciones se utilizan para rellenar arrays:

array_fill

array_fill (int $start_index, int $num, mixed $value): array
Llena un array $num veces (debe ser mayor que cero) con el valor **$value** desde la clave **$startindex**. Si **$startindex** es negativo, el primer valor tendrá como clave ese valor negativo y los demás comenzarán desde cero.

```
<?php
  $animal = array_fill(3, 2, 'perro');
  var_dump($animal);
?>
```

```
Array(2) {
  3 => string 'perro' (length=5)
  4 => string 'perro' (length=5)
}
```

```php
<?php
  $animal = array_fill(-7, 3, 'gato');
  var_dump($animal);
?>
```

```
Array(3) {
  -7 => string 'gato' (length=4)
  0 => string 'gato' (length=4)
  1 => string 'gato' (length=4)
}
```

range

range (mixed $start, mixed $end [, number $step = 1]): array
Esta función genera un array de valores en sucesión aritmética, es decir los valores desde **$start** hasta **$end** contando de paso en paso **$step** (sin superar el valor $final).

En el caso de usar letras está limitada a emplear sólo un carácter, si se emplea más de uno, sólo tiene en cuenta el primero.

```php
<?php
  $numeros = range(0, 10, 2);
  echo "<pre>";
  print_r($numeros);
  echo "</pre>";
?>
```

```
Array
(
    [0] => 0
    [1] => 2
    [2] => 4
    [3] => 6
    [4] => 8
    [5] => 10
)
```

```php
<?php
  $letras = range('a', 'd');
  echo "<pre>";
  print_r($letras);
  echo "</pre>";
?>
```

```
Array
(
    [0] => a
    [1] => b
    [2] => c
    [3] => d
)
```

Dividir arrays

Existen diferentes funciones que sirven para dividir un array en partes.

array_slice

array_slice (array $array, int $offset [, int $length = null [, bool $preserve_keys = false]]): array
Extrae una parte de un array, que comienza en **$offset** (si es negativo comienza por el final) y con una longitud de **$length**.

Si **$length** es positivo, la secuencia tendrá tantos elementos como indique el valor. Si es mayor que el propio array, se devolverán los elementos disponibles en el array. Si es negativo, la secuencia terminará en tantos elementos comenzando por el final del array. Si se omite, contendrá todos los elementos del array desde **$offset**.

El parámetro opcional **$preserve_keys** reiniciará los índices numéricos de forma predeterminada. Se pueden preservar cambiándolo a true.

```
<?php
  $entrada = array("a", "b", "c", "d", "e");

  $salida = array_slice($entrada, 2);      // "c", "d", y "e"
  $salida = array_slice($entrada, -2, 1);  // "d"
  $salida = array_slice($entrada, 0, 3);   // "a", "b", y "c"
?>
```

Observa las diferencias en las claves de los arrays.

```
<?php
  $entrada = array("a", "b", "c", "d", "e");
  var_dump($salida = array_slice($entrada, 2, -1));
  var_dump($salida = array_slice($entrada, 2, -1, true));
?>
```

```
Array(2) {
  [0]=>
  string(1) "c"
```

```
  [1]=>
  string(1) "d"
}
Array(2) {
  [2]=>
  string(1) "c"
  [3]=>
  string(1) "d"
}
```

array_splice

array_splice (array &$input, int $offset [, int $length [, mixed $replacement = array()]]): array
Elimina una porción del array y la reemplaza por algo. Comienza la eliminación desde **$offset** (si es negativo, comienza por el final).

Si se omite **$length**, se elimina todo desde **$offset** hasta el final del array. Si es positivo, se eliminan tantos elementos como indique su longitud. Si es negativo, el final de la porción eliminada será de tantos elementos como indique la longitud desde el final del array. Si es cero no se elimina ningún elemento.

Si se especifica el **array $replacemen**t, los elementos eliminados se reemplazarán por los elementos de este array.

Para eliminar todo desde **$offset** hasta el final de array cuando se ha especificado **$replacement**, se puede utilizar **count($input)** para **$length**.

```php
<?php
  $animales = array("perro", "gato", "avestruz", "oso", "toro");
  array_splice($animales, 2);
  var_dump($animales); // Devuelve perro, gato
?>
```

```
Array(2) {
  [0]=>
  string(5) "perro"
  [1]=>
  string(4) "gato"
}
```

```php
<?php
  $animales = array("perro", "gato", "avestruz", "oso", "toro");
  array_splice($animales, 1, -1);
  var_dump($animales); // Devuelve perro, toro
?>
```

```
Array(2) {
  [0]=>
  string(5) "perro"
  [1]=>
  string(4) "toro"
}
```

```
<?php
  $animales = array("perro", "gato", "avestruz", "oso", "toro");
  array_splice($animales, 1, count($animales), "rinoceronte");
  var_dump($animales); // Devuelve perro, rinoceronte
?>
```

```
Array(2) {
  [0]=>
  string(5) "perro"
  [1]=>
  string(11) "rinoceronte"
}
```

```
<?php
  $animales = array("perro", "gato", "avestruz", "oso", "toro");
  array_splice($animales, -4, 2, "ardilla");
  var_dump($animales); // Devuelve perro, ardilla, oso, toro
?>
```

```
Array(4) {
  [0]=>
  string(5) "perro"
  [1]=>
  string(7) "ardilla"
  [2]=>
  string(3) "oso"
  [3]=>
  string(4) "toro"
}
```

```
<?php
  $animales = array("perro", "gato", "avestruz", "oso", "toro");
  array_splice($animales, -4, 0, "koala");
   var_dump($animales); // Devuelve perro, koala, gato, avestruz,
oso, toro
?>
```

```
array(6) {
  [0]=>
  string(5) "perro"
  [1]=>
  string(5) "koala"
  [2]=>
  string(4) "gato"
  [3]=>
  string(8) "avestruz"
  [4]=>
  string(3) "oso"
  [5]=>
  string(4) "toro"
}
```

array_chunk

array_chunk (array $array, int $size [, bool $preserve_keys = false]): array
Divide un array en fragmentos de tamaño **$size**. El último fragmento puede contener menos elementos que size.
Devuelve un array multidimensional indexado numéricamente, empezando desde cero.

Si **$preserve_keys** es true se mantienen las claves, si no se reindexa numéricamente.

```
<?php
  $animales = array("perro", "gato", "avestruz", "oso", "toro");
  $animales2 = array_chunk($animales, 2);
  var_dump($animales2);
?>
```

```
Array(3) {
  [0]=>
  array(2) {
    [0]=>
    string(5) "perro"
    [1]=>
    string(4) "gato"
  }
  [1]=>
  array(2) {
    [0]=>
    string(8) "avestruz"
    [1]=>
    string(3) "oso"
  }
  [2]=>
```

```
  array(1) {
    [0]=>
    string(4) "toro"
  }
}
```

Modificar elementos en arrays

Las siguientes son funciones que modifican los elementos de un array.

array_shift

array_shift (array &$array): mixed
Quita el primer valor del **$array** y lo devuelve. Los arrays asociativos no varían sus claves, mientras que los numéricos son reindexados y comienzan de cero.

Esta función después de usarse ejecuta reset() en el array.

```
<?php
  $animales = array("perro", "gato", "avestruz", "oso", "toro");
  $unAnimal = array_shift($animales);
  echo "<p>$unAnimal</p>";

  var_dump($animales); // Devuelve el array $animales pero sin
"perro"
?>
```

```
<p>perro</p>
Array(4) {
  [0]=>
  string(4) "gato"
  [1]=>
  string(8) "avestruz"
  [2]=>
  string(3) "oso"
  [3]=>
  string(4) "toro"
}
```

array_unshift

array_unshift (array &$array, mixed $value1 [, mixed $...]): int
Añade los valores **$value** al inicio del array. Igual que en **array_shift()**, en los arrays asociativos no varían sus claves, mientras que los numéricos se reindexan.

```
<?php
  $animales = array("perro", "gato", "avestruz");
  $masAnimales = array_unshift($animales, "delfín", "ballena");

  echo "<p>$masAnimales</p>";

 var_dump($animales); // Devuelve el array $animales con delfín y
ballena al principio
?>
```

```
<p>5</p>
Array(5) {
  [0]=>
  string(7) "delfín"
  [1]=>
  string(7) "ballena"
  [2]=>
  string(5) "perro"
  [3]=>
  string(4) "gato"
  [4]=>
  string(8) "avestruz"
}
```

array_pop

array_pop (array &$array): mixed
Quita el último valor del array y lo devuelve.

Esta función después de usarse ejecuta reset() en el array.

```
<?php
  $animales = array("perro", "gato", "avestruz");
  $unAnimal = array_pop($animales);
  echo "<p>$unAnimal</p>";

 var_dump($animales); // Devuelve el array con perro, gato
?>
```

```
<p>avestruz</p>
Array(2) {
  [0]=>
  string(5) "perro"
  [1]=>
  string(4) "gato"
}
```

array_push

array_push (array &$array, mixed $value1 [, mixed $...]): int
Inserta uno o más elementos al final de un array. Viene a ser lo mismo que insertar un elemento a un array $array[] = valor, pero se pueden insertar varios de vez. Si se inserta sólo un valor, es mejor utilizar la forma tradicional.

```
<?php
  $animales = array("perro", "gato", "avestruz");
  $masAnimales = array_push($animales, "delfín", "ballena");

  echo "<p>$masAnimales</p>";

 var_dump($animales); // Devuelve el array $animales con delfín y
ballena al final

?>
```

```
<p>5</p>
Array(5) {
  [0]=>
  string(5) "perro"
  [1]=>
  string(4) "gato"
  [2]=>
  string(8) "avestruz"
  [3]=>
  string(7) "delfín"
  [4]=>
  string(7) "ballena"
}
```

array_flip

array_flip (array $array): array
Intercambia las claves de un array con sus valores.

Los valores del array tienen que ser válidos (de tipo integer o string), sino genera un warning y los elementos en cuestión no se incluirán en el resultado.

```
<?php
  $animales = ["perro" => "Laika", "gato" => "Garfield"];
  $array = array_flip($animales);
  var_dump($array);
?>
```

```
Array(2) {
  ["Laika"]=>
  string(5) "perro"
  ["Garfield"]=>
  string(4) "gato"
}
```

Contar elementos de un array

La función **count($array)** permite contar los elementos de un array.
int count(mixed $array_or_countable, int $mode = COUNT_NORMAL)

```
<?php
  $nombres[1] = "Ana";
  $nombres[10] = "Antonio";
  $nombres[25] = "Carmen";

  $elementos = count($nombres);

  echo "<p>El array/matriz tiene $elementos elementos.</p>";
  echo "<pre>";
  print_r($nombres);
  echo "</pre>";
?>
```

```
<p>El array/matriz tiene 3 elementos.</p>

Array
(
    [1] => Ana
    [10] => Bernardo
    [25] => Carmen
)
```

En un array multidimensional, la función **count($array)** considera el array como un vector de vectores y devuelve simplemente el número de elementos del primer nivel:

```
<?php
  $datos["pepe"]["edad"] = 25;
  $datos["pepe"]["peso"] = 80;
  $datos["juan"]["edad"] = 22;
  $datos["juan"]["peso"] = 75;
  $datos["ana"]["edad"] = 30;

  $elementos = count($datos);
```

```
  echo "<p>El array tiene $elementos elementos.</p>";
  echo"<pre>";
  print_r($datos);
  echo "</pre>";
?>
```

```
<p>El array tiene 3 elementos.</p>

Array
(
    [pepe] => Array
        (
            [edad] => 25
            [peso] => 80
        )
    [juan] => Array
        (
             [edad] => 22
             [peso] => 75
        )
     [ana] => Array
        (
             [edad] => 30
         )
)
```

Para contar todos los elementos de un array multidimensional, habría que utilizar el segundo parámetro de la función, el modo, con el valor **COUNT_RECURSIVE**.

```
<?php
  $datos["pepe"]["edad"] = 25;
  $datos["pepe"]["peso"] = 80;
  $datos["juan"]["edad"] = 22;
  $datos["juan"]["peso"] = 75;
  $datos["ana"]["edad"] = 30;

  $elementos = count($datos, COUNT_RECURSIVE);

  print "<p>El array tiene $elementos elementos.</p>";
  print "<pre>";
  print_r($datos);
  print "</pre>";
?>
```

```
<p>El array tiene 8 elementos.</p>

Array
(
    [pepe] => Array
```

```
        (
            [edad] => 25
            [peso] => 80
        )
    [juan] => Array
        (
            [edad] => 22
            [peso] => 75
        )
     [ana] => Array
        (
            [edad] => 30
         )
)
```

En el ejemplo anterior, la respuesta 8 se debe a que la función count() recursiva considera el array como un vector de vectores y cuenta los elementos que hay en cada nivel. Desde ese punto de vista, el array contiene tres elementos que a su vez contienen dos, dos y un elemento, lo que da un total de ocho elementos.
Si quisiéramos contar únicamente los elementos de un array bidimensional habría que restar los dos resultados anteriores (5 = 8 - 3):

```php
<?php
  $datos["pepe"]["edad"] = 25;
  $datos["pepe"]["peso"] = 80;
  $datos["juan"]["edad"] = 22;
  $datos["juan"]["peso"] = 75;
  $datos["ana"]["edad"] = 30;

  $elementos = count($datos, COUNT_RECURSIVE) - count($datos);

  echo "<p>El array tiene $elementos elementos.</p>";
  echo "<pre>";
  print_r($datos);
  echo "</pre>";
?>
```

```
<p>El array tiene 5 elementos.</p>

Array
(
    [pepe] => Array
        (
            [edad] => 25
            [peso] => 80
        )
    [juan] => Array
        (
            [edad] => 22
            [peso] => 75
```

```
        )
    [ana] => Array
        (
            [edad] => 30
        )
)
```

Contar cuántas veces aparece cada valor en un array

La función **array_count_values($array)** cuenta cuántos valores hay de cada valor en un array.
array_count_values(array $array): array
El array devuelto tiene como índices los valores del array original y como valores la cantidad de veces que aparece el valor.

```
<?php
  $nombres = ["Ana", "Bernardo", "Bernardo", "Ana", "Carmen",
"Ana"];

  $cuenta = array_count_values($nombres);

  echo "<pre>";
  print_r($cuenta);
  echo "</pre>";
?>
```

```
Array
(
    [Ana] => 3
    [Bernardo] => 2
    [Carmen] => 1
)
```

Máximo y mínimo

La función **max($array, ...)** devuelve el valor máximo de un array (o varias). La función **min($array, ...)** devuelve el valor mínimo de un array(o varias).
max(array $values): mixed
min(array $values): mixed

```
<?php
  $numeros = [10, 40, 15, -1];

  $maximo = max($numeros);
  $minimo = min($numeros);

  echo "<pre>";
```

```
  print_r($numeros);
  echo "</pre>";
  echo "<p>El máximo del array es $maximo.</p>";
  echo "<p>El mínimo del array es $minimo.</p>";

?>
```

```
Array
(
    [0] => 10
    [1] => 40
    [2] => 15
    [3] => -1
)

<p>El máximo del array es 40.</p>
<p>El mínimo del array es -1.</p>
```

Los valores no numéricos se tratan como 0, pero si 0 es el mínimo o el máximo, la función devuelve la cadena.

```
<?php
  $numeros = [10, 40, 15, "abc"];

  $maximo = max($numeros);
  $minimo = min($numeros);

  echo "<pre>";
  print_r($numeros);
  echo "</pre>";

  echo "<p>El máximo del array es $maximo.</p>";
  echo "<p>El mínimo del array es $minimo.</p>";
?>
```

```
Array
(
    [0] => 10
    [1] => 40
    [2] => 15
    [3] => abc
)

<p>El máximo del array es 40.</p>
<p>El mínimo del array es abc.</p>
```

Ordenar un array

Las funciones de ordenación de arrays actúan directamente en el mismo array, en vez de crear uno nuevo ordenado. Algunas de las funciones se basan en las claves del array, otras en sus valores. El orden de clasificación que llevan es: alfabético, de menor a mayor, de mayor a menor, numérico, natural, aleatorio o definido por el usuario.

Existen varias funciones para ordenar arrays. Las más simples son las siguientes:

- Cuando los índices del array que vamos a ordenar no son importantes y se pueden modificar, podemos utilizar las funciones **sort($array, $opciones)** y **rsort($array, $opciones)**, que ordenan atendiendo únicamente a los valores del array (no a sus índices), en orden creciente o decreciente, y reindexan el array:
    - **sort(array &$array, int $sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico de los valores y genera nuevos índices numéricos consecutivos a partir de cero:

```php
<?php
  $valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

  echo "<p>Array inicial:</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";

  sort($valores);

  echo "<p>Array ordenada con sort():</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";
?>
```

```
<p>Matriz inicial:</p>
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)

<p>Array ordenada con sort():</p>
Array
(
    [0] => cinco
    [1] => nueve
    [2] => uno
)
```

- **rsort(array &$array, int $sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico inverso de los valores y genera nuevos índices numéricos consecutivos a partir de cero:

```php
<?php
  $valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

  echo "<p>Array inicial:</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";

  rsort($valores);

  echo "<p>Array ordenada con rsort():</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";
?>
```

```
<p>Array inicial:</p>
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)

<p>Array ordenada con rsort():</p>
Array
(
    [0] => uno
    [1] => nueve
    [2] => cinco
)
```

- Cuando los índices del array que vamos a ordenar son importantes y no se deben modificar, podemos ordenar atendiendo únicamente a los valores del array u ordenar atendiendo únicamente a los índices del array y ordenar en orden creciente o decreciente, por lo que PHP dispone de cuatro funciones:
  - **asort(array &$array, int $sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico de los valores:

```php
<?php
  $valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

  echo "<p>Array inicial:</p>";
  echo "<pre>";
```

```
  print_r($valores);
  echo "</pre>";

  asort($valores);

  echo "<p>Array ordenada con asort():</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";
?>
```

```
<p>Array inicial:</p>
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)

<p>Array ordenada con asort():</p>
Array
(
    [5] => cinco
    [9] => nueve
    [1] => uno
)
```

- **arsort(array &$array, int $sort_flags = SORT_REGULAR): bool** ordena por orden alfabético / numérico inverso de los valores:

```
<?php
  $valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

  echo "<p>Array inicial:</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";

  arsort($valores);

  echo "<p>Array ordenada con arsort():</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";
?>
```

```
<p>Array inicial:</p>
Array
```

```
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)

<p>Array ordenada con arsort():</p>
Array
(
    [1] => uno
    [9] => nueve
    [5] => cinco
)
```

- **ksort(array &$array, int $sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico de los índices:

```
<?php
  $valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

  echo "<p>Array inicial:</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";

  ksort($valores);

  echo "<p>Array ordenada con ksort():</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";
?>
```

```
<p>Array inicial:</p>
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)

<p>Array ordenada con ksort():</p>
Array
(
    [1] => uno
    [5] => cinco
    [9] => nueve
)
```

- **krsort(array &$array, int $sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico de los índices

```php
<?php
  $valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

  echo "<p>Array inicial:</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";

  krsort($valores);

  echo "<p>Array ordenada con krsort():</p>";
  echo "<pre>";
  print_r($valores);
  echo "</pre>";
?>
```

```
<p>Array inicial:</p>
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)

<p>Array ordenada con krsort():</p>
Array
(
    [9] => nueve
    [5] => cinco
    [1] => uno
)
```

Para ver la diferencia entre estas funciones, la tabla siguiente resume los ejemplos anteriores:

Array inicial	**sort()**	**rsort()**	**asort()**	**arsort()**	**ksort()**	**krsort()**
Array ([5] => cinco [1] => uno [9] => nueve)	Array ([0] => cinco [1] => nueve [2] => uno)	Array ([0] => uno [1] => nueve [2] => cinco)	Array ([5] => cinco [9] => nueve [1] => uno)	Array ([1] => uno [9] => nueve [5] => cinco)	Array ([1] => uno [5] => cinco [9] => nueve)	Array ([9] => nueve [5] => cinco [1] => uno)

Buscar un valor en un array

La función booleana **in_array(mixed $needle, array $haystack, bool $strict = false): bool** devuelve true si el valor se encuentra en el array. Si el argumento booleano $tipo es true, in_array() comprueba además que los tipos coincidan.

```
<?php
  $valores = [10, 40, 15, -1];

  echo "<pre>";
  print_r($valores);
  echo "</pre>";

  if (in_array(15, $valores)) {
    echo "<p>15 está en el array \$valores.</p>";
  } else {
    echo "<p>15 no está en el array \$valores.</p>";
  }

  if (in_array(25, $valores)) {
    echo "<p>25 está en el array \$valores.</p>";
  } else {
    echo "<p>25 no está en el array \$valores.</p>";
  };

  if (in_array("15", $valores, true)) {
    echo "<p>\"15\" está en el array \$valores.</p>";
  } else {
    echo "<p>\"15\" no está en el array \$valores.</p>";
  }
?>
```

```
Array
(
    [0] => 10
    [1] => 40
    [2] => 15
    [3] => -1
)

<p>15 está en el array $valores.</p>
<p>25 no está en el array $valores.</p>
<p>"15" no está en el array $valores.</p>
```

La función **array_search(mixed $needle, array $haystack, bool $strict = false): mixed** busca el valor en el array y, si lo encuentra, devuelve el índice correspondiente, pero si hay varios valores coincidentes sólo devuelve el primero que encuentra.

La función **array_keys($array[, $valor[, $tipo]): array** busca el valor en el array y, si lo encuentra, devuelve un array cuyos valores son los índices de todos los elementos coincidentes.

```
<?php
  $valores = [10, 40, 15, 30, 15, 40, 15];

  echo "<pre>"; print_r($valores); echo "</pre>";

  $encontrado = array_search(15, $valores);
    echo "<p>El valor 15 se ha encontrado en la posición: $encontrado</p>";

  $encontrados = array_keys($valores, 15);
  echo "<pre>";
  print_r($encontrados);
  echo "</pre>";
?>
```

```
<p>Matriz inicial:</p>
Array
(
    [0] => 10
    [1] => 40
    [2] => 15
    [3] => 30
    [4] => 15
    [5] => 40
    [6] => 15
)

<p>El valor 15 se ha encontrado en la posición: 2</p>

Array
(
    [0] => 2
    [1] => 4
    [2] => 6
)
```

Reindexar un array

La función **array_values(array $array): array** devuelve los valores de un array en el mismo orden que en el array original, pero renumerando los índices desde cero:

```
<?php
   $nombres = ["a" => "Ana", "b" => "Bernardo", "c" => "Carmen", "d" => "David"];
```

```
  echo "<p>Array inicial:</p>";
  echo "<pre>";
  print_r($nombres);
  echo "</pre>";

  $reindexada = array_values($nombres);

  echo "<p>Array reindexada con array_values():</p>";
  echo "<pre>";
  print_r($reindexada);
  echo "</pre>";
?>
```

```
<p>Array inicial:</p>
Array
(
    [a] => Ana
    [b] => Bernardo
    [c] => Carmen
    [d] => David
)

<p>reindexada con array_values():</p>

Array
(
    [0] => Ana
    [1] => Bernardo
    [2] => Carmen
    [3] => David

)
```

Barajar los elementos de un array

La función **shuffle(array &$array): bool** baraja los valores de un array. Los índices del array original se pierden, ya que se renumeran desde cero.

```
<?php
  $numeros = [0, 1, 2, 3, 4, 5];

  echo "<p>Array inicial:</p>";
  echo "<pre>";
  print_r($numeros);
  echo "</pre>";

  shuffle($numeros);
```

```
  echo "<p>Array barajada con shuffle():</p>";
  echo "<pre>";
  print_r($numeros);
  echo "</pre>";
?>
```

```
<p>Array inicial:</p>
Array
(
    [0] => 0
    [1] => 1
    [2] => 2
    [3] => 3
    [4] => 4
    [5] => 5
)

<p>Array barajada con shuffle():</p>

Array
(
    [0] => 3
    [1] => 1
    [2] => 5
    [3] => 2
    [4] => 0
    [5] => 4
)
```

```
<?php
  $numeros = ["a" => 1, "b" => 2, "c" => 3, "d" =>4];

  echo "<p>Array inicial:</p>";
  echo "<pre>";
  print_r($numeros);
  echo "</pre>";

  shuffle($numeros);

  echo "<p>Array barajada con shuffle():</p>";
  echo "<pre>";
  print_r($numeros);
  echo "</pre>";
?>
```

```
<p>Array inicial:</p>
Array
```

```
(
    [a] => 1
    [b] => 2
    [c] => 3
    [d] => 4
)

<p>Array barajada con shuffle():</p>

Array
(
    [0] => 2
    [1] => 1
    [2] => 4
    [3] => 3
)
```

Extraer al azar un elemento de un array

La función **array_rand(array $array, int $num = 1): mixed** extrae al azar uno de los índices del array.

```
<?php
  $cuadrados = [2 => 4, 3 => 9, 7 => 49, 10 => 100];

  $indice = array_rand($cuadrados);

  echo "<pre>";
  print_r($cuadrados);
  echo "</pre>";

  echo "<p>Se ha extraído al azar el índice $indice</p>";
?>
```

```
Array
(
    [2] => 4
    [3] => 9
    [7] => 49
    [10] => 100
)

<p>Se ha extraído al azar el índice 10</p>
```

Una vez obtenido el índice se puede obtener el valor correspondiente de la matriz.

```
<?php
  $cuadrados = [2 => 4, 3 => 9, 7 => 49, 10 => 100];
```

```
  $indice = array_rand($cuadrados);

  echo "<pre>";
  print_r($cuadrados);
  echo "</pre>";

     echo "<p>Con el índice $indice extraemos el valor
$cuadrados[$indice]</p>";
?>
```

```
Array
(
    [2] => 4
    [3] => 9
    [7] => 49
    [10] => 100
)

<p>Con el índice 3 extraemos el valor 9</p>
```

Eliminar valores repetidos

La función **array_unique(array $array, int $sort_flags = SORT_STRING): array)** devuelve un array en el que se han eliminado los valores repetidos.

```
<?php
  $inicial = [0, 0, 1, 3, 1, 3, 2, 1];

  $final = array_unique($inicial);

  echo "<pre>";
  print_r($final);
  echo "</pre>";
?>
```

```
Array
(
    [0] => 0
    [2] => 1
    [3] => 3
    [6] => 2
)
```