array_sum »
« array_slice

- Manual de PHP
- Referencia de funciones
- Extensiones relacionadas con variable y tipo
- Arrays
- Funciones de Arrays

Change language: Spanish

Submit a Pull Request Report a Bug

array_splice

(PHP 4, PHP 5, PHP 7, PHP 8)

array_splice — Elimina una porción del array y la reemplaza con otra cosa

Descripción ¶

array_splice(
  array `&$input`,
  int `$offset`,
  int `$length` = 0,
  mixed `$replacement` = array()
): array

Elimina los elementos designados por `offset` y `length` del array `input`, y los reemplaza con los elementos del array `replacement`, si se proporcionan.

Observe que las claves numéricas de `input` no se preservan.

> **Nota**: Si `replacement` no es un array, será moldeado a uno (esto es, `(array) $replacement`). Esto puede resultar en un comportamiento inesperado cuando se usa un objeto o `replacement` **`null`**

Parámetros ¶

`input`

> El array de entrada.

`offset`

> Si el índice dado por `offset` es positivo, el inicio de la porción eliminada estará en ese índice desde el principio del array `input`. Si `offset` es negativo, se comienza desde el final del array `input`.

`length`

> Si se omite la longitud dada por `length`, se elimina todo desde `offset` hasta el final del array. Si se especifica `length` y es positivo, se eliminarán tantos elementos como indique la longitud. Si se especifica `length` y es negativo, el final de la porción eliminada será de tantos elementos como indique la longitud desde el final del array. Si se especifica `length` y es cero, no se eliminará ningún elemento. Consejo: para eliminar todo desde `offset` hasta el final del array cuando también se especifique `replacement`, use `count($input)` para `length`.

`replacement`

Si se especifica el array `replacement`, los elementos eliminados serán reemplazados con los elementos de este array.

Si `offset` y `length` son tales que no se elimina nada, los elementos del array `replacement` serán insertados en el lugar especificado por `offset`. Observe que las claves del array replacement no se preservan.

Si `replacement` es sólo un elemento, no es necesario poner `array()` alrededor de él, a menos que el elemento sea un array, un objeto o **`null`**.

Valores devueltos ¶

Devuelve un array que consiste en los elementos extraídos.

Ejemplos ¶

Ejemplo #1 Ejemplos de array_splice()

```php
<?php
$entrada = array("rojo", "verde", "azul", "amarillo");
array_splice($entrada, 2);
// $entrada ahora es array("rojo", "verde")

$entrada = array("rojo", "verde", "azul", "amarillo");
array_splice($entrada, 1, -1);
// $entrada ahora es array("rojo", "amarillo")

$entrada = array("rojo", "verde", "azul", "amarillo");
array_splice($entrada, 1, count($entrada), "naranja");
// $entrada ahora es array("rojo", "naranja")

$entrada = array("rojo", "verde", "azul", "amarillo");
array_splice($entrada, -1, 1, array("negro", "granate"));
// $entrada ahora es array("rojo", "verde",
//          "azul", "negro", "granate")

$entrada = array("rojo", "verde", "azul", "amarillo");
array_splice($entrada, 3, 0, "púpura");
// $entrada ahora es array("rojo", "verde",
//          "azul", "púpura", "amarillo");
?>
```

Ejemplo #2 Ejemplos de array_splice()

Las siguientes sentencias cambian el valor de *$entrada* de la misma manera:

```php
<?php
array_push($entrada, $x, $y);
array_splice($entrada, count($entrada), 0, array($x, $y));
array_pop($entrada);
array_splice($entrada, -1);
array_shift($entrada);
array_splice($entrada, 0, 1);
array_unshift($entrada, $x, $y);
array_splice($entrada, 0, 0, array($x, $y));
$entrada[$x] = $y; // para arrays donde la clave es igual al índice
```

```
array_splice($entrada, $x, 1, $y);
?>
```

Ver también ¶

- array_slice() - Extraer una parte de un array
- unset() - Destruye una o más variables especificadas
- array_merge() - Combina dos o más arrays

+ add a note

User Contributed Notes 39 notes

up
down
37
mrsohailkhan at gmail dot com ¶
11 years ago

array_splice, split an array into 2 arrays. The returned arrays is the 2nd argument actually and the used array e.g $input here contains the 1st argument of array, e.g

```
<?php
$input = array("red", "green", "blue", "yellow");
print_r(array_splice($input, 3)); // Array ( [0] => yellow )
print_r($input); //Array ( [0] => red [1] => green [2] => blue )
?>
```

if you want to replace any array value do simple like that,

first search the array index you want to replace

```
<?php $index = array_search('green', $input);// index = 1 ?>
```

and then use it as according to the definition

```
<?php
array_splice($input, $index, 1, array('mygreeen')); //Array ( [0] => red [1] => mygreeen [2] => blue [3] => yellow )
?>
```

so here green is replaced by mygreen.

here 1 in array_splice above represent the number of items to be replaced. so here start at index '1' and replaced only one item which is 'green'

up
down
14
StanE ¶
7 years ago

array_splice() does not preserve numeric keys. The function posted by "weikard at gmx dot de" won't do that either because array_merge() does not preserve numeric keys either.

Use following function instead:

```
<?php
function arrayInsert($array, $position, $insertArray)
{
```

```
    $ret = [];

    if ($position == count($array)) {
        $ret = $array + $insertArray;
    }
    else {
        $i = 0;
        foreach ($array as $key => $value) {
            if ($position == $i++) {
                $ret += $insertArray;
            }

            $ret[$key] = $value;
        }
    }

    return $ret;
}
?>

Example:
<?php
$a = [
    295 => "Hello",
    58 => "world",
];

$a = arrayInsert($a, 1, [123 => "little"]);

/*
Output:
Array
(
    [295] => Hello
    [123] => little
    [58] => world
)
*/
?>
```

It preserves numeric keys. Note that the function does not use a reference to the original array but returns a new array (I see absolutely no reason how the performance would be increased by using a reference when modifying an array through PHP script code).

up
down
22
royanee at yahoo dot com ¶
9 years ago

When trying to splice an associative array into another, array_splice is missing two key ingredients:

- a string key for identifying the offset
- the ability to preserve keys in the replacement array

This is primarily useful when you want to replace an item in an array with another item, but want to maintain the ordering of the array without rebuilding the array one entry at a time.

```
<?php
function array_splice_assoc(&$input, $offset, $length, $replacement) {
```

```
        $replacement = (array) $replacement;
        $key_indices = array_flip(array_keys($input));
        if (isset($input[$offset]) && is_string($offset)) {
                $offset = $key_indices[$offset];
        }
        if (isset($input[$length]) && is_string($length)) {
                $length = $key_indices[$length] - $offset;
        }

        $input = array_slice($input, 0, $offset, TRUE)
                + $replacement
                + array_slice($input, $offset + $length, NULL, TRUE);
}

$fruit = array(
        'orange' => 'orange',
        'lemon' => 'yellow',
        'lime' => 'green',
        'grape' => 'purple',
        'cherry' => 'red',
);

// Replace lemon and lime with apple
array_splice_assoc($fruit, 'lemon', 'grape', array('apple' => 'red'));

// Replace cherry with strawberry
array_splice_assoc($fruit, 'cherry', 1, array('strawberry' => 'red'));
?>
```

Note: I have not tested this with negative offsets and lengths.

up
down
17

daniele centamore ¶

13 years ago

just useful functions to move an element using array_splice.

```
<?php

// info at danielecentamore dot com

// $input  (Array) - the array containing the element
// $index (int) - the index of the element you need to move

function moveUp($input,$index) {
      $new_array = $input;

       if((count($new_array)>$index) && ($index>0)){
                array_splice($new_array, $index-1, 0, $input[$index]);
                array_splice($new_array, $index+1, 1);
             }

       return $new_array;
}

function moveDown($input,$index) {
       $new_array = $input;
```

```
        if(count($new_array)>$index) {
                  array_splice($new_array, $index+2, 0, $input[$index]);
                  array_splice($new_array, $index, 1);
              }

        return $new_array;
}

$input = array("red", "green", "blue", "yellow");

$newinput = moveUp($input, 2);
// $newinput is array("red", "blue", "green", "yellow")

$input = moveDown($newinput, 1);
// $input is array("red", "green", "blue", "yellow")

?>
```

up
down
6
gilberg_vrn ¶
6 years ago

array_splice with preserve keys

```
<?php
function array_splice_preserve_keys(&$array, $from, $length = null) {
    $result = array_slice($array, $from, $length, true);
    $array = array_slice($array, $from + $length, null, true);

    return $result;
}
?>
```

Example:

```
<?php
$array = [
    1 => 'a',
    2 => 'b',
    26 => 'z'
];

var_dump(array_splice_preserve_keys($array, 0, 1), $array);
/**
* array(1) {
*  [1]=>
*   string(1) "a"
*  }
* array(2) {
*  [2]=>
*  string(1) "b"
*  [26]=>
*  string(1) "z"
* }
*/
?>
```

up
down

7

charette dot s at gmail ¶

12 years ago

If you want to append null values wrap them in an array:

```
<?php

    $a = array('Hey', 'hey', 'my', 'my');
    array_splice($a, 1, 0, null);
    print_r($a);

?>
Array
(
    [0] => Hey
    [1] => hey
    [2] => my
    [3] => my
)

<?php

    $b = array('Hey', 'hey', 'my', 'my');
    array_splice($b, 1, 0, array(null));
    print_r($b);

?>
Array
(
    [0] => Hey
    [1] =>
    [2] => hey
    [3] => my
    [4] => my
)
```

up

down

7

gideon at i6developments dot com ¶

18 years ago

array_splice dynamically updates the total number of entries into the array. So for instance I had a case where I needed to insert a value into every 4th entry of the array from the back. The problem was when it added the first, because the total number was dynamically updated, it would only add after the 3rd then the 2nd and so one. The solution I found is to track the number of inserts which were done and account for them dynamically.

Code:

```
<?php
    $modarray = array_reverse($mili);
    $trig=1;
        foreach($modarray as $rubber => $glue) {
            if($rubber!="<BR>") {
            $i++;
            $b++;
            if ($i==4) {
                $trig++;
                if($trig<=2) {
                array_splice($modarray,$b,0,"<BR>");
```

```
                }elseif($trig>=3){
                array_splice($modarray,$b+($trig-2),0,"<BR>");
                }
                $i=0;
                };
                };
        };
    $fixarray = array_reverse($modarray);

?>
```

up
down
19

weikard at gmx dot de ¶

17 years ago

You cannot insert with array_splice an array with your own key. array_splice will always insert it with the key "0".

```
<?php
// [DATA]
$test_array = array (
  row1 => array (col1 => 'foobar!', col2 => 'foobar!'),
  row2 => array (col1 => 'foobar!', col2 => 'foobar!'),
  row3 => array (col1 => 'foobar!', col2 => 'foobar!')
);

// [ACTION]
array_splice ($test_array, 2, 0, array ('rowX' => array ('colX' => 'foobar2')));
echo '<pre>'; print_r ($test_array); echo '</pre>';
?>

[RESULT]

Array (
    [row1] => Array (
            [col1] => foobar!
            [col2] => foobar!
        )

    [row2] => Array (
            [col1] => foobar!
            [col2] => foobar!
        )

    [0] => Array (
            [colX] => foobar2
        )

    [row3] => Array (
            [col1] => foobar!
            [col2] => foobar!
        )
)

But you can use the following function:

function array_insert (&$array, $position, $insert_array) {
  $first_array = array_splice ($array, 0, $position);
```

```
  $array = array_merge ($first_array, $insert_array, $array);
}

<?php
// [ACTION]

array_insert ($test_array, 2, array ('rowX' => array ('colX' => 'foobar2')));
echo '<pre>'; print_r ($test_array); echo '</pre>';
?>

[RESULT]

Array (
    [row1] => Array (
            [col1] => foobar!
            [col2] => foobar!
        )

    [row2] => Array (
            [col1] => foobar!
            [col2] => foobar!
        )

    [rowX] => Array (
            [colX] => foobar2
        )

    [row3] => Array (
            [col1] => foobar!
            [col2] => foobar!
        )
)

[NOTE]

The position "0" will insert the array in the first position (like array_shift). If you try a
position higher than the langth of the array, you add it to the array like the function
array_push.
```

up
down
4

kbrown at horizon dot sk dot ca ¶

19 years ago

```
[ Editor's Note: If you're not concerned with the indexes being contiguously numbered (such as for
an associative array) then unset($ar[$ind]); will accomplish the same as the code below without
requiring splice/splice/merge.  If contiguous numbering IS a concern (such as for indexed arrays),
you can still save time by using: unset($ar[$ind]); $ar = array_values($ar); ]

Removing elements from arrays

This works better - much quicker

<?php
$ar = array("einstein", "bert", "colin", "descartes", "renoir");
$a = array_slice($ar, 0, $ind);
$b = array_slice($ar, $ind + 1);
$ar = array_merge($a, $b);
?>
```

up
down
5
plintus at smtp dot ru ¶
19 years ago
key-safe:

```
<?php
function array_kslice ($array, $offset, $length = 0) {
$k = array_slice (array_keys ($array), $offset, $length);
$v = array_slice (array_values ($array), $offset, $length);
for ($i = 0; $i < count ($k); $i ++) $r[$k[$i]] = $v[$i];
return $r;
}
?>
```

smth like this. hope you like it more than versions above :)

up
down
7
news_yodpeirs at thoftware dot de ¶
12 years ago
Splicing with NULL as replacement may result in unexpected behavior too. Typecasting NULL into an array results in an empty array (as "(array)NULL" equals "array()"). That means, instead of creating an element with value NULL just no new element ist created (just as if there was no replacement specified).

If you want the splicing to create a new element with value NULL you have to use "array(NULL)" instead of NULL.

You should expect this if you read the explanation carefully, but just as objects are considered as a special case for replacement, NULL should be too.

The explanation of replacement better should read: "If replacement is just one element it is not necessary to put array() around it, unless the element is an array itself, an object or NULL."

And the note better should be: "If replacement is not an array, it will be typecast to one (i.e. (array) $parameter). This may result in unexpected behavior when using an object or NULL replacement."

jmtc

up
down
4
csaba at alum dot mit dot edu ¶
17 years ago
Appending arrays
If you have an array $a2 whose values you would like to append to an array $a1 then four methods you could use are listed below in order of increasing time. The last two methods took significantly more time than the first two. The most surprising lesson is that using the & incurs a time hit.

```
<?php
foreach ($a2 as $elem) $a1[]=$elem;
foreach ($a2 as &$elem) $a1[]=$elem;
array_splice ($a1, count($a1), 0, $a2);
$a1 = array_merge($a1, $a2);
?>
```

Csaba Gabor from Vienna
up
down
3
guillaume dot lacourt at gmail dot com ¶
7 years ago
Using array_splice when you traverse array with internal pointer's function reset the array, eg:

```
<?php
end($arrOfData);
$last = key($arrOfData);
reset($arrOfData);
while (($data = current($arrOfData))) {
  if ($last === key($arrOfData)) {
    array_splice($arrOfData, $last, 1);
   // current($arrOfData) => first value of $arrOfData
  }
}
```

up
down
3
thom ¶
8 years ago
Maybe it will help someone else: I was trying to strip off the last part of an array using this section, more or less as follows:

```
<?php array_splice($array, $offset); ?>
```

Now it could occur in my code that <?php $offset === 0 ?>, in which case the array is returned as-is and not, as you might expect, an empty array because everything is stripped off. Obviously it is not really useful anyway to "strip off everything", but I was reminded of that the hard way and this may spare someone some time, hopefully.
up
down
3
jrhardytwothousandtwo at yahoo dot com ¶
20 years ago
A reference is made to INSERT'ing into an array here with array_splice, however its not explained very well. I hope this example will help others find what took me days to research.

```
<?php
$original_array = array(1,2,3,4,5);
$insert_into_key_position = 3;
$item_to_insert = "blue";

$returned = array_splice($original_array, $insert_into_key_position, 0, $item_to_insert);

// $original_array will now show:

// 1,2,3,blue,4,5
?>
```

Remember that you are telling the array to insert the element into the KEY position. Thus the elements start with key 0 and so on 0=>1, 1=>2, 2=>3, 3=>blue, 4=>4, 5=>5. And walla, you've inserted. I can't say if this is of any value for named keys, or multidimensional arrays. However it does work for single dimensional arrays.

$returned should be an empty array as nothing was returned. This would have substance if you were doing a replace instead.

up
down
1

dead dot screamer at seznam dot cz ¶

13 years ago

I need <?php array_Splice()?> function, that use array keys instead of order (offset and length) because of associated arrays, and this is result:

```
<?php
/**
*    first variation
*
*    $input is input array
*    $start is index of slice begin
*    $end is index of slice end, if this is null, $replacement will be inserted (in the same way as original array_Slice())
*indexes of $replacement are preserved in both examples
*/
function array_KSplice1(&$input, $start, $end=null, $replacement=null)
{
    $keys=array_Keys($input);
    $values=array_Values($input);
    if($replacement!==null)
    {
        $replacement=(array)$replacement;
        $rKeys=array_Keys($replacement);
        $rValues=array_Values($replacement);
    }

    $start=array_Search($start,$keys,true);
    if($start===false)
        return false;
    if($end!==null)
    {
        $end=array_Search($end,$keys,true);
        // if $end not found, exit
        if($end===false)
            return false;
        // if $end is before $start, exit
        if($end<$start)
            return false;
        // index to length
        $end-=$start-1;
    }

    // optional arguments
    if($replacement!==null)
    {
        array_Splice($keys,$start,$end,$rKeys);
        array_Splice($values,$start,$end,$rValues);
    }
    else
    {
        array_Splice($keys,$start,$end);
        array_Splice($values,$start,$end);
    }
```

```
    $input=array_Combine($keys,$values);

    return $input;
}

/**
*    second variation
*
*    $input is input array
*    $start is index of slice begin
*    $length is length of slice, what will be replaced, if is zero, $replacement will be inserted
(in the same way as original array_Slice())
*/
function array_KSplice2(&$input, $start, $length=0, $replacement=null)
{
    $keys=array_Keys($input);
    $values=array_Values($input);
    if($replacement!==null)
    {
        $replacement=(array)$replacement;
        $rKeys=array_Keys($replacement);
        $rValues=array_Values($replacement);
    }

    $start=array_Search($start,$keys,true);
    if($start===false)
        return false;

    // optional arguments
    if($replacement!==null)
    {
        array_Splice($keys,$start,$length,$rKeys);
        array_Splice($values,$start,$length,$rValues);
    }
    else
    {
        array_Splice($keys,$start,$length);
        array_Splice($values,$start,$length);
    }

    $input=array_Combine($keys,$values);

    return $input;
}

$array=range(1,10);
var_Dump(array_KSplice1($array,3,3,array(100=>101,102,103,104)));

$array=range(1,10);
var_Dump(array_KSplice2($array,3,3,array(100=>101,102,103,104)));

?>

Both examples output:
array(11) {
  [0]=>
  int(1)
```

```
  [1]=>
  int(2)
  [2]=>
  int(3)
  [100]=>
  int(101)
  [101]=>
  int(102)
  [102]=>
  int(103)
  [103]=>
  int(104)
  [6]=>
  int(7)
  [7]=>
  int(8)
  [8]=>
  int(9)
  [9]=>
  int(10)
}
```

up
down
1

pauljamescampbell at gmail dot com ¶
14 years ago

Here's my own take on an array slice method that preserves keys from an associative array.

```
<?php
/**
* Array slice function that preserves associative keys
*
* @function associativeArraySlice
*
* @param  Array  $array  Array to slice
* @param  Integer  $start
* @param  Integer  $end
*
* @return  Array
*/
function associativeArraySlice($array, $start, $end) {
    // Method param restrictions
    if($start < 0) $start = 0;
    if($end > count($array)) $end = count($array);

    // Process vars
    $new = Array();
    $i = 0;

    // Loop
    foreach($array as $key => $value) {
        if($i >= $start && $i < $end) {
            $new[$key] = $value;
        }
        $i++;
    }
    return($new);
```

```
}
?>
```

up
down
3

ahigerd at stratitec dot com ¶

15 years ago

A comment on array_merge mentioned that array_splice is faster than array_merge for inserting values. This may be the case, but if your goal is instead to reindex a numeric array, array_values() is the function of choice. Performing the following functions in a 100,000-iteration loop gave me the following times: ($b is a 3-element array)

```
array_splice($b, count($b)) => 0.410652
$b = array_splice($b, 0) => 0.272513
array_splice($b, 3) => 0.26529
$b = array_merge($b) => 0.233582
$b = array_values($b) => 0.151298
```

up
down
2

Paul ¶

16 years ago

In PHP 4.3.10, at least, it seems that elements that are inserted as part of the replacement array are inserted BY REFERENCE (that is, as though with the =& rather than = assignment operation). So if your replacement array contains elements that references to variables that you can also access via other variable name, then this will be true of the elements in the final array too.

In particular, this means that it is safe to use array_splice() on arrays of objects, as you won't be creating copies of the objects (as it is so easy to do in PHP 4).

up
down
2

Anonymous ¶

20 years ago

Please note that array_splice() 's second argument is an OFFSET and not an INDEX.

Lets say you want to

```
$array_of_items = array ('nothing','myitem','hisitem','heritem');
$sid = array_search('myitem',$array_of_items);
echo $sid; /* prints out 1, since index element 1 is "myitem" */
```

Now, lets say we want to remove that "myitem" from the array:

```
<?php
$array_of_items = array_splice($array_of_items,(1+$sid),1);
?>
```

Notice how you have to add a one to the $sid variable? That is because offset item 1 is "nothing" and since $sid is currently 1 (the index of "myitem"), we add 1 more to it to find out its OFFSET.

DO NOT DO THIS:

```
$array_of_items = array_splice($array_of_items,$sid,1);
```

up
down
1

paule at cs dot tamu dot edu ¶

20 years ago

to kokos@lac.lviv.ua:

Good point about the code not doing what you expected.

The failure to check for the insert case like you pointed out is not a bug, however. I didn't add code to handle that because the key of such an added index is more or less undefined in an unordered associative array. Put another way, if your array is associative and not auto-indexed, you most likely care enough about your keys to want to set them explicitly.

up
down
1

paule at cs dot tamu dot edu ¶

20 years ago

After reading KoKos' post above, I thought that the code I posted right before his should do what he wanted. However, my original post neglected to note the little "Tip" in the documentation above, about a single element replacement.

If one changes the lines in my code above that says:

```
<?php
                if(is_array($replacement))
                    foreach($replacement as $r_key=>$r_value)
                        $new_array[$r_key]=$r_value;
?>
```

to instead say:

```
<?php
                if(is_string($replacement))
                    $new_array[$key]=$replacement;
                elseif(is_array($replacement))
                    foreach($replacement as $r_key=>$r_value)
                        $new_array[$r_key]=$r_value;
?>
```

that will solve the problem.

Sorry for the omission.

up
down
1

mip at ycn dot com ¶

15 years ago

Ever wounder what array_splice is doing to your references, then try this little script and see the output.

```
<?php

$a = "a";
$b = "b";
$c = "c";
$d = "d";
$arr = array();
$arr[] =& $a;
$arr[] =& $b;
$arr[] =& $c;
array_splice($arr,1,0,array($d));
$sec_arr = array();
```

```
$sec_arr[] =& $d;
array_splice($arr,1,0,$sec_arr);

$arr[0] = "test"; // should be $a
$arr[3] = "test2"; // should be $b
$arr[1] = "this be d?"; // should be $d
$arr[2] = "or this be d?"; // should be $d
var_dump($arr);
var_dump($a);
var_dump($b);
var_dump($d);
?>

The output will be (PHP 4.3.3):

array(5) {
  [0]=>
  &string(4) "test"
  [1]=>
  &string(10) "this be d?"
  [2]=>
  string(13) "or this be d?"
  [3]=>
  &string(5) "test2"
  [4]=>
  &string(1) "c"
}
string(4) "test"
string(5) "test2"
string(10) "this be d?"

So array_splice is reference safe, but you have to be careful about the generation of the
replacement array.

have fun, cheers!
```

up
down
1

bdjumakov at gmail dot com ¶

16 years ago

```
Someone might find this function usefull. It just takes a given element from the array and moves
it before given element into the same array.

<?php
function array_move($which, $where, $array)
{
    $tmp  = array_splice($array, $which, 1);
    array_splice($array, $where, 0, $tmp);
    return $array;
}
?>
```

up
down
1

kokos at lac dot lviv dot ua ¶

20 years ago

```
It may seem obvious from the above posts, but cost me a bit of
braindamage to figure this out...
```

Contrary to the equivalence noted on this page
$input[$x] = $y <==> array_splice ($input, $x, 1, $y)
array_splice() will not always work as expected,
even provided that you have only INTEGER keys!

The following code:

```
   $t=array('a','b','c','d','e');
   var_dump($t);

<?php
   unset($t[0],$t[1],$t[3]);
   $t[0]='f';
   var_dump($t);

   array_splice($t,0,1,'g');
   var_dump($t);
?>
```

Will produce:

```
array(5) {
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [2]=>
  string(1) "c"
  [3]=>
  string(1) "d"
  [4]=>
  string(1) "e"
}
array(3) {
  [2]=>
  string(1) "c"
  [4]=>
  string(1) "e"
  [0]=>
  string(1) "f"
}
array(3) {
  [0]=>
  string(1) "g"
  [1]=>
  string(1) "e"
  [2]=>
  string(1) "f"
}
```

Note the position of $t[0] in the second call to var_dump().
And of course, array_splice() left it intact, changing $t[2] instead.
This is because it operates the _offset_, not the _index_. :)
I think that "equivalence note" should be considered buggy. ;)))

Best wishes.
KoKos.

up
down

2

Anonymous ¶

1 year ago

the following:

```
        $input = [[5=>"richard=red"], [15=>"york=yellow"], [25=>"gave=green"],
[30=>"battle=blue"], [35=>"in=indigo"], [40=>"vain=violet"]];
        array_splice($input, 2, 0, [[10=>"of=orange"]]);
        var_dump($input);
```

gives this:

```
array (size=7)
  0 =>
    array (size=1)
      5 => string 'richard=red' (length=11)
  1 =>
    array (size=1)
      15 => string 'york=yellow' (length=11)
  2 =>
    array (size=1)
      10 => string 'of=orange' (length=9)
  3 =>
    array (size=1)
      25 => string 'gave=green' (length=10)
  4 =>
    array (size=1)
      30 => string 'battle=blue' (length=11)
  5 =>
    array (size=1)
      35 => string 'in=indigo' (length=9)
  6 =>
    array (size=1)
      40 => string 'vain=violet' (length=11)
```

up
down

2

vitospericolato at gmail dot com ¶

6 years ago

To remove elements from an array, based on array values:

```
<?php
$i_to_remove=array();

foreach($array_to_prune as $i=>$value){
  if(cond_to_delete($value)) $i_to_remove[]=$i;
}
foreach($i_to_remove as $j=>$i)
  array_splice($array_to_prune,$i-$j,1);

?>
```

up
down

0

Hayley Watson ¶

5 years ago

For an analogous function that works on strings rather than arrays, see substr_replace.

up

down
2
randomdestination at gmail dot com ¶
17 years ago
To split an associative array based on it's keys, use this function:

```
<?php
function &array_split(&$in) {
    $keys = func_get_args();
    array_shift($keys);

    $out = array();
    foreach($keys as $key) {
        if(isset($in[$key]))
            $out[$key] = $in[$key];
        else
            $out[$key] = null;
        unset($in[$key]);
    }

    return $out;
}
?>
```

Example:

```
<?php
$testin = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4);
$testout =& array_split($testin, 'a', 'b', 'c');

print_r($testin);
print_r($testout);
?>
```

Will print:

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

Hope this helps anyone!

up
down
0
antrik ¶
8 years ago
Prompted by dire need, and inspired by some of the existing notes, I came up with this:

```
/* Like array_splice(), but preserves the key(s) of the replacement array. */
function array_splice_assoc(&$input, $offset, $length = 0, $replacement = array()) {
  $tail = array_splice($input, $offset);
  $extracted = array_splice($tail, 0, $length);
```

```
  $input += $replacement + $tail;
  return $extracted;
};
```

Apart from preserving the keys, it behaves just like the regular array_splice() for all cases I could think of.

So for example the regular array_splice()

```
$input = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5, 'f' =>6);
print_r(array_splice($input, -4, 3, array('foo1' => 'bar', 'foo2' => 'baz')));
print_r($input);
```

will give:

```
Array
(
    [c] => 3
    [d] => 4
    [e] => 5
)
Array
(
    [a] => 1
    [b] => 2
    [0] => bar
    [1] => baz
    [f] => 6
)
```

But with array_splice_assoc()

```
$input = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5, 'f' =>6);
print_r(array_splice_assoc($input, -4, 3, array('foo1' => 'bar', 'foo2' => 'baz')));
print_r($input);
```

we get:

```
Array
(
    [c] => 3
    [d] => 4
    [e] => 5
)
Array
(
    [a] => 1
    [b] => 2
    [foo1] => bar
    [foo2] => baz
    [f] => 6
)
```

A typical use case would be replacing an element identified by a particular key, which we could achieve with:

```
$input = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5, 'f' =>6);
array_splice_assoc($input, array_search('d', array_keys($input)), 1, array('foo' => 'bar'));
```

```
print_r($input);

giving us:

Array
(
    [a] => 1
    [b] => 2
    [c] => 3
    [foo] => bar
    [e] => 5
    [f] => 6
)
```

up
down
0

news_yodpeirs at thoftware dot de ¶
11 years ago

Sometimes you may want to insert one array into another and just work on with the resulting array. array_splice() doesn't support this, as the resulting array isn't the returned value but the first argument, which is changed by reference.

Therefore you may use the following function, which inserts array $ins in array $src at position $pos. $rep can be used if $ins shouldn't be just inserted, but should replace some existing elements (the number of elements to be replaced is given in $rep).

```
<?php
function array_insert($src,$ins,$pos,$rep=0) {
  array_splice($src,$pos,$rep,$ins);
  return($src);
}
?>
```

up
down
-1

madmax at max-worlds dot net ¶
14 years ago

Note: If replacement is not an array, it will be typecast to one (i.e. (array) $parameter). This may result in unexpected behavior when using an object replacement .

Example :

```
<?php
class A()
{
    private $a;
    private $b;
    public function __construct()
    {
        $this->a = "foo";
        $this->b = "bar";
    }
}

$array = array();
array_splice($array, 0, 0, new A());
print_r($array);
?>
```

```
Outputs :

Array : Array
{
    [0] => foo
    [1] => bar
}

Solution : Enforce the array() on the object.

<?php
array_splice($array, 0, 0, array(new Object()));
?>

Source : http://bugs.php.net/bug.php?id=44485
```

up
down
-1
Anonymous ¶
8 years ago

```
<?php
function array_slice2( $array, $offset, $length = 0 )
{
  if( $offset < 0 )
    $offset = sizeof( $array ) + $offset;

  $length = ( !$length ? sizeof( $array ) : ( $length < 0 ? sizeof( $array ) - $length : $length +
$offset ) );

  for( $i = $offset; $i < $length; $i++ )
    $tmp[] = $array[$i];

  return $tmp;
}
?>
```

up
down
-3
tsunaquake DOESNTLIKESPAM @ wp DOT pl ¶
20 years ago

```
It is possible to use a string instead of offset, eg if you want to deletre the entry
$myArray['entry'] then you can simply do it like this:

<?php
array_splice($myArray, 'entry', 1);
?>

Note that you can use unset($myArray['entry']) as well but then, it doesn't enable you to remove
more than one entry and it doesn't replace anything in the array, if that's what you intend to do.
```

up
down
-4
leingang AT math DOT rutgers DOT edu ¶
20 years ago

```
array_splice resets the internal pointer of $input.  In fact, many array functions do this.
Caveat programmor!
```

up

down
-3
loushou - life dot 42 at gmail dot com ¶
14 years ago
i miss posted the actual function...
here is the real one lol

```
<?php

function q_sort(&$Info, $Index, $Left, $Right)
{
  echo "memory usage <b>".memory_get_usage()."</b><br/>\n";
  $L_hold = $Left;
  $R_hold = $Right;
  $Pivot = $Left;
  $PivotValue = $Info[$Left];
  while ($Left < $Right)
  {
    while (($Info[$Right][$Index] >= $PivotValue[$Index]) && ($Left < $Right))
      $Right--;
    if ($Left != $Right)
    {
      $Info[$Left] = $Info[$Right];
      $Left++;
    }
    while (($Info[$Left][$Index] <= $PivotValue[$Index]) && ($Left < $Right))
      $Left++;
    if ($Left != $Right)
    {
      $Info[$Right] = $Info[$Left];
      $Right--;
    }
  }
  $Info[$Left] = $PivotValue;
  $Pivot = $Left;
  $Left = $L_hold;
  $Right = $R_hold;
  if ($Left < $Pivot)
    q_sort($Info, $Index, $Left, $Pivot-1);
  if ($Right > $Pivot)
    q_sort($Info, $Index, $Pivot+1, $Right);
}

?>
```

up
down
-3
rolandfoxx at yahoo dot com ¶
18 years ago
Be careful, array_splice does not behave like you might expect should you try to pass it an object as the replacement argument. Consider the following:

```
<?php
//Very truncated
class Tree {
  var $childNodes

  function addChild($offset, $node) {
```

```
    array_splice($this->childNodes, $offset, 0, $node);
    //...rest of function
  }

}

class Node {
  var $stuff
  ...
}

$tree = new Tree();
// ...set 2 nodes using other functions...
echo (count($tree->childNodes)); //Gives 2
$newNode = new Node();
// ...set node attributes here...
$tree->addChild(1, $newNode);
echo(count($tree->childNodes)); //Expect 3?  wrong!
?>
```

In this case, the array has a number of items added to it equal to the number of attributes in the new Node object and the values thereof I.e, if your Node object has 2 attributes with values "foo" and "bar", count($tree->childNodes) will now return 4, with the items "foo" and "bar" added to it. I'm not sure if this qualifies as a bug, or is just a byproduct of how PHP handles objects.

Here's a workaround for this problem:

```
function array_insertobj(&$array, $offset, $insert) {
  $firstPart = array_slice($array, 0, $offset);
  $secondPart = array_slice($array, $offset);
  $insertPart = array($insert);
  $array = array_merge($firstPart, $insertPart, $secondPart);
}
```

Note that this function makes no allowances for when $offset equals the first or last index in the array. That's because array_unshift and array_push work just fine in those cases. It's only array_splice that can trip you up. Obviously, this is kinda tailor-made for arrays with numeric keys when you don't really care what said keys are, but i'm sure you could adapt it for associative arrays if you needed it.

up
down
-7

Francis ¶

14 years ago

Do you need to sort a 2D array on just one of its variables while trying to preserve somewhat the original order?

```
<?php

function sort_2d_array($array, $position, $order = "ASC"){
  if (!is_array($array)) return $array;
  if (count($array) < 2) return $array;
  $new = array($array[0]);
  for ($cnt = 1; $cnt <= count($array) - 1; $cnt++){
    $stop = 0;
    $splice = 0;
    for ($newcnt = 0; $newcnt <= count($new) - 1; $newcnt++){
      if ($stop == 0){
```

```
        if ($order == "ASC")
        if ($array[$cnt][$position] < $new[$newcnt][$position]){
          $splice = $newcnt;
          $stop = 1;
        } // splice position for ASC
        if ($order == "DESC")
        if ($array[$cnt][$position] > $new[$newcnt][$position]){
          $splice = $newcnt;
          $stop = 1;
        } // splice position for DESC
      } // stop vying for position
    } // cycle through new array to find position
    if ($stop == 0){
      $new[] = $array[$cnt];
    } else {
      array_splice($new, $splice, 0, array($array[$cnt]));
    } // splice into new array while keeping somewhat the original order
  } // cycle through original array
  return $new;
} // sort_2d_array

?>
```

Application Example: In-House Search Engine
Here we are trying to find the word apple in the website by sort of the most recent occurances first, but the number of occurances first.

We've already sorted the mysql output by the date desc and have counted the no of occurances and have placed those in an array for the final query.

I've used this function to further sort the occurances but somewhat keep the original mysql sort order.

```
Key
[0] Record number
     [0] Record ID
     [1] Source Table
     [2] No of Occurances Pinged

  ---------------------------

[0]
     [0] 24530
     [1] Blogs
     [2] 1

[1]
     [0] 24400
     [1] Blogs
     [2] 1

[2]
     [0] 24240
     [1] Blogs
     [2] 4

[3]
     [0] 243422
```

```
     [1] Classifieds
     [2] 1

[4]
     [0] 243100
     [1] Classifieds
     [2] 1

After running...

<?php
sort_2d_array($array, 2, "DESC");
?>

We have...

[0]
     [0] 24240
     [1] Blogs
     [2] 4

[1]
     [0] 24530
     [1] Blogs
     [2] 1

[2]
     [0] 24400
     [1] Blogs
     [2] 1

[3]
     [0] 243422
     [1] Classifieds
     [2] 1

[4]
     [0] 243100
     [1] Classifieds
     [2] 1

Might be useful to someone...
```

up
down
-6
strata_ranger at hotmail dot com ¶
13 years ago

Should you want a similar function for splicing strings together, here is a rough equivalent:

```
<?php
function str_splice($input, $offset, $length=null, $splice='')
{
  $input = (string)$input;
  $splice = (string)$splice;
  $count = strlen($input);

  // Offset handling (negative values measure from end of string)
  if ($offset<0) $offset = $count + $offset;
```

```
  // Length handling (positive values measure from $offset; negative, from end of string; omitted
= end of string)
  if (is_null($length)) $length = $count;
  elseif ($length < 0)  $length = $count-$offset+$length;

  // Do the splice
  return substr($input, 0, $offset) . $splice . substr($input, $offset+$length);
}

$string = "The fox jumped over the lazy dog.";

// Outputs "The quick brown fox jumped over the lazy dog."
echo str_splice($string, 4, 0, "quick brown ");

?>
```

Obviously this is not for cases where all you need to do is a simple search-and-replace.

+ add a note

- Funciones de Arrays
  - array_change_key_case
  - array_chunk
  - array_column
  - array_combine
  - array_count_values
  - array_diff_assoc
  - array_diff_key
  - array_diff_uassoc
  - array_diff_ukey
  - array_diff
  - array_fill_keys
  - array_fill
  - array_filter
  - array_flip
  - array_intersect_assoc
  - array_intersect_key
  - array_intersect_uassoc
  - array_intersect_ukey
  - array_intersect
  - array_is_list
  - array_key_exists
  - array_key_first
  - array_key_last
  - array_keys
  - array_map
  - array_merge_recursive
  - array_merge
  - array_multisort
  - array_pad
  - array_pop
  - array_product
  - array_push
  - array_rand
  - array_reduce
  - array_replace_recursive
  - array_replace
  - array_reverse