Arrays

Introduction

These functions allow you to interact with and manipulate arrays in various ways. Arrays are essential for storing, managing, and operating on sets of variables.

Simple and multi-dimensional arrays are supported, and may be either user created or created by another function. There are specific database handling functions for populating arrays from database queries, and several functions return arrays.

Please see the Arrays section of the manual for a detailed explanation of how arrays are implemented and used in PHP. See also Array operators for other ways how to manipulate the arrays.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

There is no installation needed to use these functions; they are part of the PHP core.

Runtime Configuration

This extension has no configuration directives defined in *php.ini*.

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are always available as part of the PHP core.

CASE_LOWER (integer)
CASE_LOWER is used with array_change_key_case() and is used to convert array keys to lower case. This is also the default case for array_change_key_case().

CASE_UPPER (integer)
CASE_UPPER is used with array_change_key_case() and is used to convert array keys to upper case.

Sorting order flags:
SORT_ASC (integer)
SORT_ASC is used with array_multisort() to sort in ascending order.

SORT_DESC (integer)
SORT_DESC is used with array_multisort() to sort in descending order.

Sorting type flags: used by various sort functions
SORT_REGULAR (integer)
SORT_REGULAR is used to compare items normally.

SORT_NUMERIC (integer)
SORT_NUMERIC is used to compare items numerically.

SORT_STRING (integer)
SORT_STRING is used to compare items as strings.

SORT_LOCALE_STRING (integer)
SORT_LOCALE_STRING is used to compare items as strings, based on the current locale. Added in PHP 4.4.0 and 5.0.2.

COUNT_NORMAL (integer)

COUNT_RECURSIVE (integer)

EXTR_OVERWRITE (integer)

EXTR_SKIP (integer)

EXTR_PREFIX_SAME (integer)

EXTR_PREFIX_ALL (integer)

EXTR_PREFIX_INVALID (integer)

EXTR_PREFIX_IF_EXISTS (integer)

EXTR_IF_EXISTS (integer)

EXTR_REFS (integer)

Array Functions

See Also

See also is_array(), explode(), implode(), split(), preg_split(), and unset().

array_change_key_case

array_change_key_case -- Changes all keys in an array

Description

array **array_change_key_case** (array `$input` [, int `$case`])

Returns an array with all keys from *`input`* lowercased or uppercased. Numbered indices are left as is.

Parameters

`input`
The array to work on

`case`
Either **CASE_UPPER** or **CASE_LOWER** (default)

Return Values

Returns an array with its keys lower or uppercased, or false if *`input`* is not an array.

Errors/Exceptions

Throws **E_WARNING** if *`input`* is not an array.

Examples

Example #1 - array_change_key_case() example

```
<?php
$input_array = array("FirSt" => 1, "SecOnd" => 4);
print_r(array_change_key_case($input_array, CASE_UPPER));
?>
```

The above example will output:

```
Array
(
   [FIRST] => 1
   [SECOND] => 4
)
```

Notes

Note
If an array has indices that will be the same once run through this function (e.g. "keY" and "kEY"), the value that is later in the array will override other indices.

array_chunk

array_chunk -- Split an array into chunks

Description

array **array_chunk** (array `$input`, int `$size` [, bool `$preserve_keys`])

Chunks an array into *`size`* large chunks. The last chunk may contain less than *`size`* elements.

Parameters

`input`
: The array to work on

`size`
: The size of each chunk

`preserve_keys`
: When set to **TRUE** keys will be preserved. Default is **FALSE** which will reindex the chunk numerically

Return Values

Returns a multidimensional numerically indexed array, starting with zero, with each dimension containing *`size`* elements.

Errors/Exceptions

If *`size`* is less than 1 **E_WARNING** will be thrown and **NULL** returned.

Examples

Example #2 - array_chunk() example

```
<?php
$input_array = array('a', 'b', 'c', 'd', 'e');
print_r(array_chunk($input_array, 2));
print_r(array_chunk($input_array, 2, true));
?>
```

The above example will output:

```
Array
```

```
(
   [0] => Array
       (
           [0] => a
           [1] => b
       )

   [1] => Array
       (
           [0] => c
           [1] => d
       )

   [2] => Array
       (
           [0] => e
       )

)
Array
(
   [0] => Array
       (
           [0] => a
           [1] => b
       )

   [1] => Array
       (
           [2] => c
           [3] => d
       )

   [2] => Array
       (
           [4] => e
       )

)
```

array_combine

array_combine -- Creates an array by using one array for keys and another for its values

Description

array **array_combine** (array `$keys`, array `$values`)

Creates an array by using the values from the *`keys`* array as keys and the values from the *`values`* array as the corresponding values.

Parameters

`keys`
Array of keys to be used. Illegal values for key will be converted to string.

`values`
Array of values to be used

Return Values

Returns the combined array, **FALSE** if the number of elements for each array isn't equal or if the arrays are empty.

Errors/Exceptions

Throws **E_WARNING** if *`keys`* and *`values`* are either empty or the number of elements does not match.

Examples

Example #3 - A simple array_combine() example

```
<?php
$a = array('green', 'red', 'yellow');
$b = array('avocado', 'apple', 'banana');
$c = array_combine($a, $b);

print_r($c);
?>
```

The above example will output:

```
Array
(
    [green]  => avocado
```

```
    [red]    => apple
    [yellow] => banana
)
```

See Also

- array_merge()
- array_walk()
- array_values()

array_count_values

array_count_values -- Counts all the values of an array

Description

array **array_count_values** (array `$input`)

array_count_values() returns an array using the values of the *`input`* array as keys and their frequency in *`input`* as values.

Parameters

`input`
The array of values to count

Return Values

Returns an associative array of values from *`input`* as keys and their count as value.

Errors/Exceptions

Throws **E_WARNING** for every element which is not string or integer.

Examples

Example #4 - array_count_values() example

```
<?php
$array = array(1, "hello", 1, "world", "hello");
print_r(array_count_values($array));
?>
```

The above example will output:

```
Array
(
   [1] => 2
   [hello] => 2
   [world] => 1
)
```

See Also

- count()
- array_unique()
- array_values()
- count_chars()

array_diff_assoc

array_diff_assoc -- Computes the difference of arrays with additional index check

Description

array **array_diff_assoc** (array `$array1`, array `$array2` [, array `$...`])

Compares *`array1`* against *`array2`* and returns the difference. Unlike array_diff() the array keys are used in the comparision.

Parameters

`array1`
The array to compare from

`array2`
An array to compare against

`...`
More arrays to compare against

Return Values

Returns an array containing all the values from *`array1`* that are not present in any of the other arrays.

Examples

Example #5 - array_diff_assoc() example

In this example you see the *"a" => "green"* pair is present in both arrays and thus it is not in the ouput from the function. Unlike this, the pair *0 => "red"* is in the ouput because in the second argument *"red"* has key which is *1*.

```
<?php
$array1 = array("a" => "green", "b" => "brown", "c" => "blue", "red");
$array2 = array("a" => "green", "yellow", "red");
$result = array_diff_assoc($array1, $array2);
print_r($result);
?>
```

The above example will output:

```
Array
(
```

```
    [b] => brown
    [c] => blue
    [0] => red
)
```

Example #6 - array_diff_assoc() **example**

Two values from *key* => *value* pairs are considered equal only if *(string) $elem1 === (string) $elem2*. In other words a strict check takes place so the string representations must be the same.

```
<?php
$array1 = array(0, 1, 2);
$array2 = array("00", "01", "2");
$result = array_diff_assoc($array1, $array2);
print_r($result);
?>
```

The above example will output:

```
Array
(
    [0] => 0
    [1] => 1
    )
```

Notes

Note

This function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using, for example, *array_diff_assoc($array1[0], $array2[0]);*.

See Also

- array_diff()
- array_intersect()
- array_intersect_assoc()

array_diff_key

array_diff_key -- Computes the difference of arrays using keys for comparison

Description

array **array_diff_key** (array `$array1`, array `$array2` [, array `$...`])

Compares the keys from *`array1`* against the keys from *`array2`* and returns the difference. This function is like array_diff() except the comparison is done on the keys instead of the values.

Parameters

`array1`
The array to compare from

`array2`
An array to compare against

`...`
More arrays to compare against

Return Values

Returns an array containing all the entries from *`array1`* that are not present in any of the other arrays.

Examples

Example #7 - array_diff_key() example

The two keys from the *key => value* pairs are considered equal only if *(string) $key1 === (string) $key2*. In other words a strict type check is executed so the string representation must be the same.

```
<?php
$array1 = array('blue'  => 1, 'red'  => 2, 'green'  => 3, 'purple' => 4);
$array2 = array('green' => 5, 'blue' => 6, 'yellow' => 7, 'cyan'   => 8);

var_dump(array_diff_key($array1, $array2));
?>
```

The above example will output:

```
array(2) {
```

```
  ["red"]=>
  int(2)
  ["purple"]=>
  int(4)
}
```

Notes

Note
This function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using *array_diff_key($array1[0], $array2[0]);*.

See Also

- array_diff()
- array_udiff()
- array_diff_assoc()
- array_diff_uassoc()
- array_udiff_assoc()
- array_udiff_uassoc()
- array_diff_ukey()
- array_intersect()
- array_intersect_assoc()
- array_intersect_uassoc()
- array_intersect_key()
- array_intersect_ukey()

array_diff_uassoc

array_diff_uassoc -- Computes the difference of arrays with additional index check which is performed by a user supplied callback function

Description

array **array_diff_uassoc** (array `$array1`, array `$array2` [, array `$...`], callback `$key_compare_func`)

Compares *`array1`* against *`array2`* and returns the difference. Unlike array_diff() the array keys are used in the comparision.

Unlike array_diff_assoc() an user supplied callback function is used for the indices comparision, not internal function.

Parameters

`array1`
: The array to compare from

`array2`
: An array to compare against

`...`
: More arrays to compare against

`key_compare_func`
: callback function to use. The callback function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Return Values

Returns an array containing all the entries from *`array1`* that are not present in any of the other arrays.

Examples

Example #8 - array_diff_uassoc() example
The *"a" => "green"* pair is present in both arrays and thus it is not in the ouput from the function. Unlike this, the pair *0 => "red"* is in the ouput because in the second argument *"red"* has key which is *1*.

```php
<?php
function key_compare_func($a, $b)
{
   if ($a === $b) {
       return 0;
   }
   return ($a > $b)? 1:-1;
}

$array1 = array("a" => "green", "b" => "brown", "c" => "blue", "red");
$array2 = array("a" => "green", "yellow", "red");
$result = array_diff_uassoc($array1, $array2, "key_compare_func");
print_r($result);
?>
```

The above example will output:

```
Array
(
   [b] => brown
   [c] => blue
   [0] => red
)
```

The equality of 2 indices is checked by the user supplied callback function.

Notes

Note
This function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using, for example, *array_diff_uassoc($array1[0], $array2[0], "key_compare_func");*.

See Also

- array_diff()
- array_diff_assoc()
- array_udiff()
- array_udiff_assoc()
- array_udiff_uassoc()
- array_intersect()
- array_intersect_assoc()
- array_uintersect()
- array_uintersect_assoc()
- array_uintersect_uassoc()

array_diff_ukey

array_diff_ukey -- Computes the difference of arrays using a callback function on the keys for comparison

Description

array **array_diff_ukey** (array `$array1`, array `$array2` [, array `$...`], callback `$key_compare_func`)

Compares the keys from *`array1`* against the keys from *`array2`* and returns the difference. This function is like array_diff() except the comparison is done on the keys instead of the values.

Unlike array_diff_key() an user supplied callback function is used for the indices comparision, not internal function.

Parameters

`array1`
The array to compare from

`array2`
An array to compare against

`...`
More arrays to compare against

`key_compare_func`
callback function to use. The callback function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Return Values

Returns an array containing all the entries from *`array1`* that are not present in any of the other arrays.

Examples

Example #9 - array_diff_ukey() example

```
<?php
function key_compare_func($key1, $key2)
{
```

```
    if ($key1 == $key2)
        return 0;
    else if ($key1 > $key2)
        return 1;
    else
        return -1;
}

$array1 = array('blue'  => 1, 'red'  => 2, 'green'  => 3, 'purple' => 4);
$array2 = array('green' => 5, 'blue' => 6, 'yellow' => 7, 'cyan'   => 8);

var_dump(array_diff_ukey($array1, $array2, 'key_compare_func'));
?>
```

The above example will output:

```
array(2) {
 ["red"]=>
 int(2)
 ["purple"]=>
 int(4)
}
```

Notes

Note

This function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using *array_diff_ukey($array1[0], $array2[0], 'callback_func');*.

See Also

- array_diff()
- array_udiff()
- array_diff_assoc()
- array_diff_uassoc()
- array_udiff_assoc()
- array_udiff_uassoc()
- array_diff_key()
- array_intersect()
- array_intersect_assoc()
- array_intersect_uassoc()
- array_intersect_key()
- array_intersect_ukey()

array_diff

array_diff -- Computes the difference of arrays

Description

array **array_diff** (array `$array1`, array `$array2` [, array `$...`])

Compares *`array1`* against *`array2`* and returns the difference.

Parameters

`array1`
The array to compare from

`array2`
An array to compare against

`...`
More arrays to compare against

Return Values

Returns an array containing all the entries from *`array1`* that are not present in any of the other arrays.

Examples

Example #10 - array_diff() example

```
<?php
$array1 = array("a" => "green", "red", "blue", "red");
$array2 = array("b" => "green", "yellow", "red");
$result = array_diff($array1, $array2);

print_r($result);
?>
```

Multiple occurrences in $array1 are all treated the same way. This will output :

```
Array
(
   [1] => blue
)
```

Notes

Note
Two elements are considered equal if and only if *(string) $elem1 === (string) $elem2*. In words: when the string representation is the same.

Note
This function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using *array_diff($array1[0], $array2[0]);*.

Warning
This was broken in PHP 4.0.4!

See Also

- array_diff_assoc()
- array_intersect()
- array_intersect_assoc()

array_fill_keys

array_fill_keys -- Fill an array with values, specifying keys

Description

array **array_fill_keys** (array `$keys`, mixed `$value`)

Fills an array with the value of the *`value`* parameter, using the values of the *`keys`* array as keys.

Parameters

`keys`
: Array of values that will be used as keys. Illegal values for key will be converted to string.

`value`
: Value to use for filling

Return Values

Returns the filled array

Examples

Example #11 - array_fill_keys() example

```
<?php
$keys = array('foo', 5, 10, 'bar');
$a = array_fill_keys($keys, 'banana');
print_r($a);
?>
```

The above example will output:

```
Array
(
   [foo] => banana
   [5] => banana
   [10] => banana
   [bar] => banana
)
```

See Also

- array_fill()
- array_combine()

array_fill

array_fill -- Fill an array with values

Description

array **array_fill** (int `$start_index`, int `$num`, mixed `$value`)

Fills an array with *num* entries of the value of the *value* parameter, keys starting at the *start_index* parameter.

Parameters

start_index
: The first index of the returned array

num
: Number of elements to insert

value
: Value to use for filling

Return Values

Returns the filled array

Errors/Exceptions

Throws a **E_WARNING** if *num* is less than one.

Examples

Example #12 - array_fill() example

```
<?php
$a = array_fill(5, 6, 'banana');
$b = array_fill(-2, 2, 'pear');
print_r($a);
print_r($b);
?>
```

The above example will output:

```
Array
(
    [5]  => banana
```

```
    [6]  => banana
    [7]  => banana
    [8]  => banana
    [9]  => banana
    [10] => banana
)
Array
(
    [-2] => pear
    [0] => pear
)
```

Notes

See also the Arrays section of manual for a detailed explanation of negative keys.

See Also

- str_repeat()
- range()

array_filter

array_filter -- Filters elements of an array using a callback function

Description

array **array_filter** (array `$input` [, callback `$callback`])

Iterates over each value in the *`input`* array passing them to the *`callback`* function. If the *`callback`* function returns true, the current value from *`input`* is returned into the result array. Array keys are preserved.

Parameters

`input`
The array to iterate over

`callback`
The callback function to use If no *`callback`* is supplied, all entries of *`input`* equal to **FALSE** (see converting to boolean) will be removed.

Return Values

Returns the filtered array.

Examples

Example #13 - array_filter() example

```
<?php
function odd($var)
{
   return($var & 1);
}

function even($var)
{
   return(!($var & 1));
}

$array1 = array("a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5);
$array2 = array(6, 7, 8, 9, 10, 11, 12);

echo "Odd :\n";
print_r(array_filter($array1, "odd"));
echo "Even:\n";
print_r(array_filter($array2, "even"));
```

```
?>
```

The above example will output:

```
Odd :
Array
(
   [a] => 1
   [c] => 3
   [e] => 5
)
Even:
Array
(
   [0] => 6
   [2] => 8
   [4] => 10
   [6] => 12
)
```

Example #14 - array_filter() without *callback*

```
<?php

$entry = array(
            0 => 'foo',
            1 => false,
            2 => -1,
            3 => null,
            4 => ''
         );

print_r(array_filter($entry));
?>
```

The above example will output:

```
Array
(
   [0] => foo
   [2] => -1
)
```

Notes

Caution

If the array is changed from the callback function (e.g. element added, deleted or unset) the behavior of this function is undefined.

See Also

- array_map()
- array_reduce()
- array_walk()

array_flip

array_flip -- Exchanges all keys with their associated values in an array

Description

array **array_flip** (array `$trans`)

array_flip() returns an array in flip order, i.e. keys from *`trans`* become values and values from *`trans`* become keys.

Note that the values of *`trans`* need to be valid keys, i.e. they need to be either integer or string. A warning will be emitted if a value has the wrong type, and the key/value pair in question *will not be flipped*.

If a value has several occurrences, the latest key will be used as its values, and all others will be lost.

Parameters

`trans`
An array of key/value pairs to be flipped.

Return Values

Returns the flipped array on success and **FALSE** on failure.

Examples

Example #15 - array_flip() example

```
<?php
$trans = array_flip($trans);
$original = strtr($str, $trans);
?>
```

Example #16 - array_flip() example : collision

```
<?php
$trans = array("a" => 1, "b" => 1, "c" => 2);
$trans = array_flip($trans);
print_r($trans);
```

```
?>
```

now *$trans* is:

```
Array
(
    [1] => b
    [2] => c
)
```

See Also

- array_values()
- array_keys()
- array_reverse()

array_intersect_assoc

array_intersect_assoc -- Computes the intersection of arrays with additional index check

Description

array **array_intersect_assoc** (array `$array1`, array `$array2` [, array `$...`])

array_intersect_assoc() returns an array containing all the values of *`array1`* that are present in all the arguments. Note that the keys are used in the comparison unlike in array_intersect().

Parameters

`array1`
The array with master values to check.

`array2`
An array to compare values against.

`array`
A variable list of arrays to compare.

Return Values

Returns an associative array containing all the values in *`array1`* that are present in all of the arguments.

Examples

Example #17 - array_intersect_assoc() example

```
<?php
$array1 = array("a" => "green", "b" => "brown", "c" => "blue", "red");
$array2 = array("a" => "green", "yellow", "red");
$result_array = array_intersect_assoc($array1, $array2);
print_r($result_array);
?>
```

The above example will output:

```
Array
(
   [a] => green
)
```

In our example you see that only the pair *"a" => "green"* is present in both arrays and thus is returned. The value *"red"* is not returned because in *$array1* its key is *0* while the key of "red" in *$array2* is *1*.

The two values from the *key => value* pairs are considered equal only if *(string) $elem1 === (string) $elem2*. In other words a strict type check is executed so the string representation must be the same.

See Also

- array_intersect()
- array_uintersect_assoc()
- array_intersect_uassoc()
- array_uintersect_uassoc()
- array_diff()
- array_diff_assoc()

array_intersect_key

array_intersect_key -- Computes the intersection of arrays using keys for comparison

Description

array **array_intersect_key** (array `$array1`, array `$array2` [, array `$...`])

array_intersect_key() returns an array containing all the values of *`array1`* which have matching keys that are present in all the arguments.

Parameters

`array1`
The array with master keys to check.

`array2`
An array to compare keys against.

`array`
A variable list of arrays to compare.

Return Values

Returns an associative array containing all the values of *`array1`* which have matching keys that are present in all arguments.

Examples

Example #18 - array_intersect_key() example

```
<?php
$array1 = array('blue'  => 1, 'red'  => 2, 'green'  => 3, 'purple' => 4);
$array2 = array('green' => 5, 'blue' => 6, 'yellow' => 7, 'cyan'   => 8);

var_dump(array_intersect_key($array1, $array2));
?>
```

The above example will output:

```
array(2) {
 ["blue"]=>
 int(1)
 ["green"]=>
 int(3)
}
```

In our example you see that only the keys *'blue'* and *'green'* are present in both arrays and thus returned. Also notice that the values for the keys *'blue'* and *'green'* differ between the two arrays. A match still occurs because only the keys are checked. The values returned are those of `array1`.

The two keys from the *key => value* pairs are considered equal only if *(string) $key1 === (string) $key2*. In other words a strict type check is executed so the string representation must be the same.

See Also

- array_diff()
- array_udiff()
- array_diff_assoc()
- array_diff_uassoc()
- array_udiff_assoc()
- array_udiff_uassoc()
- array_diff_key()
- array_diff_ukey()
- array_intersect()
- array_intersect_assoc()
- array_intersect_uassoc()
- array_intersect_ukey()

array_intersect_uassoc

array_intersect_uassoc -- Computes the intersection of arrays with additional index check, compares indexes by a callback function

Description

array **array_intersect_uassoc** (array `$array1`, array `$array2` [, array `$...`], callback `$key_compare_func`)

array_intersect_uassoc() returns an array containing all the values of *`array1`* that are present in all the arguments. Note that the keys are used in the comparison unlike in array_intersect().

The index comparison is done by a user supplied callback function. It must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Parameters

`array1`
Initial array for comparision of the arrays.

`array2`
First array to compare keys against.

`array`
Variable list of array arguments to compare values against.

`key_compare_func`
User supplied callback function to do the comparision.

Return Values

Returns the values of *`array1`* whose values exist in all of the arguments.

Examples

Example #19 - array_intersect_uassoc() example

```
<?php
$array1 = array("a" => "green", "b" => "brown", "c" => "blue", "red");
$array2 = array("a" => "GREEN", "B" => "brown", "yellow", "red");

print_r(array_intersect_uassoc($array1, $array2, "strcasecmp"));
```

```
?>
```

The above example will output:

```
Array
(
    [b] => brown
)
```

See Also

- array_intersect()
- array_intersect_assoc()
- array_uintersect_assoc()
- array_uintersect_uassoc()
- array_intersect_key()
- array_intersect_ukey()

array_intersect_ukey

array_intersect_ukey -- Computes the intersection of arrays using a callback function on the keys for comparison

Description

array **array_intersect_ukey** (array `$array1`, array `$array2` [, array `$...`], callback `$key_compare_func`)

array_intersect_ukey() returns an array containing all the values of *`array1`* which have matching keys that are present in all the arguments.

This comparison is done by a user supplied callback function. It must return an integer less than, equal to, or greater than zero if the first key is considered to be respectively less than, equal to, or greater than the second.

Parameters

`array1`
: Initial array for comparision of the arrays.

`array2`
: First array to compare keys against.

`array`
: Variable list of array arguments to compare keys against.

`key_compare_func`
: User supplied callback function to do the comparision.

Return Values

Returns the values of *`array1`* whose keys exist in all the arguments.

Examples

Example #20 - array_intersect_ukey() example

```
<?php
function key_compare_func($key1, $key2)
{
   if ($key1 == $key2)
       return 0;
   else if ($key1 > $key2)
       return 1;
```

```
    else
        return -1;
}

$array1 = array('blue'  => 1, 'red'  => 2, 'green'  => 3, 'purple' => 4);
$array2 = array('green' => 5, 'blue' => 6, 'yellow' => 7, 'cyan'   => 8);

var_dump(array_intersect_ukey($array1, $array2, 'key_compare_func'));
?>
```

The above example will output:

```
array(2) {
 ["blue"]=>
 int(1)
 ["green"]=>
 int(3)
}
```

In our example you see that only the keys *'blue'* and *'green'* are present in both arrays and thus returned. Also notice that the values for the keys *'blue'* and *'green'* differ between the two arrays. A match still occurs because only the keys are checked. The values returned are those of `array1`.

See Also

- array_diff()
- array_udiff()
- array_diff_assoc()
- array_diff_uassoc()
- array_udiff_assoc()
- array_udiff_uassoc()
- array_diff_key()
- array_diff_ukey()
- array_intersect()
- array_intersect_assoc()
- array_intersect_uassoc()
- array_intersect_key()

array_intersect

array_intersect -- Computes the intersection of arrays

Description

array **array_intersect** (array `$array1`, array `$array2` [, array `$...`])

array_intersect() returns an array containing all the values of *`array1`* that are present in all the arguments. Note that keys are preserved.

Parameters

`array1`
The array with master values to check.

`array2`
An array to compare values against.

`array`
A variable list of arrays to compare.

Return Values

Returns an array containing all of the values in *`array1`* whose values exist in all of the parameters.

Examples

Example #21 - array_intersect() example

```
<?php
$array1 = array("a" => "green", "red", "blue");
$array2 = array("b" => "green", "yellow", "red");
$result = array_intersect($array1, $array2);
print_r($result);
?>
```

The above example will output:

```
Array
(
   [a] => green
   [0] => red
)
```

Notes

Note
Two elements are considered equal if and only if *(string) $elem1 === (string) $elem2*. In words: when the string representation is the same.

See Also

- array_intersect_assoc()
- array_diff()
- array_diff_assoc()

array_key_exists

array_key_exists -- Checks if the given key or index exists in the array

Description

bool **array_key_exists** (mixed `$key`, array `$search`)

array_key_exists() returns **TRUE** if the given *key* is set in the array. *key* can be any value possible for an array index. array_key_exists() also works on objects.

Parameters

key
Value to check.

search
An array with keys to check.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #22 - array_key_exists() example

```
<?php
$search_array = array('first' => 1, 'second' => 4);
if (array_key_exists('first', $search_array)) {
   echo "The 'first' element is in the array";
}
?>
```

Note

The name of this function is **key_exists()** in PHP 4.0.6.

Example #23 - array_key_exists() **vs** isset()

isset() does not return **TRUE** for array keys that correspond to a **NULL** value, while array_key_exists() does.

```
<?php
$search_array = array('first' => null, 'second' => 4);

// returns false
isset($search_array['first']);

// returns true
array_key_exists('first', $search_array);
?>
```

See Also

- isset()
- array_keys()
- in_array()

array_keys

array_keys -- Return all the keys of an array

Description

array **array_keys** (array `$input` [, mixed `$search_value` [, bool `$strict`]])

array_keys() returns the keys, numeric and string, from the *`input`* array.

If the optional *`search_value`* is specified, then only the keys for that value are returned. Otherwise, all the keys from the *`input`* are returned. As of PHP 5, you can use *`strict`* parameter for comparison including type (===).

Parameters

`input`
An array containing keys to return.

`search_value`
If specified, then only keys containing these values are returned.

`strict`
As of PHP 5, this parameter determines if strict comparision (===) should be used during the search.

Return Values

Returns an array of all the keys in *`input`*.

Examples

Example #24 - array_keys() example

```
<?php
$array = array(0 => 100, "color" => "red");
print_r(array_keys($array));

$array = array("blue", "red", "green", "blue", "blue");
print_r(array_keys($array, "blue"));

$array = array("color" => array("blue", "red", "green"),
               "size"  => array("small", "medium", "large"));
print_r(array_keys($array));
?>
```

The above example will output:

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

See Also

- array_values()
- array_key_exists()

array_map

array_map -- Applies the callback to the elements of the given arrays

Description

array **array_map** (callback `$callback`, array `$arr1` [, array `$...`])

array_map() returns an array containing all the elements of *`arr1`* after applying the *`callback`* function to each one. The number of parameters that the *`callback`* function accepts should match the number of arrays passed to the array_map()

Parameters

`callback`
Callback function to run for each element in each array.

`arr1`
An array to run through the *`callback`* function.

`array`
Variable list of array arguments to run through the *`callback`* function.

Return Values

Returns an array containing all the elements of *`arr1`* after applying the *`callback`* function to each one.

Examples

Example #25 - array_map() example

```
<?php
function cube($n)
{
   return($n * $n * $n);
}

$a = array(1, 2, 3, 4, 5);
$b = array_map("cube", $a);
print_r($b);
?>
```

This makes *$b* have:

```
Array
```

```
(
   [0] => 1
   [1] => 8
   [2] => 27
   [3] => 64
   [4] => 125
)
```

Examples

Example #26 - array_map() - using more arrays

```
<?php
function show_Spanish($n, $m)
{
   return("The number $n is called $m in Spanish");
}

function map_Spanish($n, $m)
{
   return(array($n => $m));
}

$a = array(1, 2, 3, 4, 5);
$b = array("uno", "dos", "tres", "cuatro", "cinco");

$c = array_map("show_Spanish", $a, $b);
print_r($c);

$d = array_map("map_Spanish", $a , $b);
print_r($d);
?>
```

The above example will output:

```
// printout of $c
Array
(
   [0] => The number 1 is called uno in Spanish
   [1] => The number 2 is called dos in Spanish
   [2] => The number 3 is called tres in Spanish
   [3] => The number 4 is called cuatro in Spanish
   [4] => The number 5 is called cinco in Spanish
)

// printout of $d
Array
(
   [0] => Array
       (
           [1] => uno
       )

   [1] => Array
```

```
        (
            [2] => dos
        )

    [2] => Array
        (
            [3] => tres
        )

    [3] => Array
        (
            [4] => cuatro
        )

    [4] => Array
        (
            [5] => cinco
        )

)
```

Usually when using two or more arrays, they should be of equal length because the callback function is applied in parallel to the corresponding elements. If the arrays are of unequal length, the shortest one will be extended with empty elements.

An interesting use of this function is to construct an array of arrays, which can be easily performed by using **NULL** as the name of the callback function

Example #27 - Creating an array of arrays

```
<?php
$a = array(1, 2, 3, 4, 5);
$b = array("one", "two", "three", "four", "five");
$c = array("uno", "dos", "tres", "cuatro", "cinco");

$d = array_map(null, $a, $b, $c);
print_r($d);
?>
```

The above example will output:

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => one
            [2] => uno
        )

    [1] => Array
        (
            [0] => 2
            [1] => two
```

```
            [2] => dos
        )

    [2] => Array
        (
            [0] => 3
            [1] => three
            [2] => tres
        )

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

    [4] => Array
        (
            [0] => 5
            [1] => five
            [2] => cinco
        )

)
```

If the array argument contains string keys then the returned array will contain string keys if and only if exactly one array is passed. If more than one argument is passed then the returned array always has integer keys.

Example #28 - array_map() - with string keys

```
<?php
$arr = array("stringkey" => "value");
function cb1($a) {
   return array ($a);
}
function cb2($a, $b) {
   return array ($a, $b);
}
var_dump(array_map("cb1", $arr));
var_dump(array_map("cb2", $arr, $arr));
var_dump(array_map(null,  $arr));
var_dump(array_map(null, $arr, $arr));
?>
```

The above example will output:

```
array(1) {
 ["stringkey"]=>
 array(1) {
   [0]=>
   string(5) "value"
 }
```

```
}
array(1) {
 [0]=>
 array(2) {
   [0]=>
   string(5) "value"
   [1]=>
   string(5) "value"
 }
}
array(1) {
 ["stringkey"]=>
 string(5) "value"
}
array(1) {
 [0]=>
 array(2) {
   [0]=>
   string(5) "value"
   [1]=>
   string(5) "value"
 }
}
```

See Also

- array_filter()
- array_reduce()
- array_walk()
- create_function()

information about the callback type

array_merge_recursive

array_merge_recursive -- Merge two or more arrays recursively

Description

array **array_merge_recursive** (array `$array1` [, array `$...`])

array_merge_recursive() merges the elements of one or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.

If the input arrays have the same string keys, then the values for these keys are merged together into an array, and this is done recursively, so that if one of the values is an array itself, the function will merge it with a corresponding entry in another array too. If, however, the arrays have the same numeric key, the later value will not overwrite the original value, but will be appended.

Parameters

array1
: Initial array to merge.

array
: Variable list of arrays to recursively merge.

Return Values

An array of values resulted from merging the arguments together.

Examples

Example #29 - array_merge_recursive() example

```
<?php
$ar1 = array("color" => array("favorite" => "red"), 5);
$ar2 = array(10, "color" => array("favorite" => "green", "blue"));
$result = array_merge_recursive($ar1, $ar2);
print_r($result);
?>
```

The above example will output:

```
Array
(
   [color] => Array
       (
```

```
            [favorite] => Array
                (
                    [0] => red
                    [1] => green
                )

            [0] => blue
        )

    [0] => 5
    [1] => 10
)
```

See Also

- array_merge()

array_merge

array_merge -- Merge one or more arrays

Description

array **array_merge** (array `$array1` [, array `$array2` [, array `$...`]])

Merges the elements of one or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.

If the input arrays have the same string keys, then the later value for that key will overwrite the previous one. If, however, the arrays contain numeric keys, the later value will *not* overwrite the original value, but will be appended.

If only one array is given and the array is numerically indexed, the keys get reindexed in a continuous way.

Parameters

`array1`
Initial array to merge.

`array`
Variable list of arrays to recursively merge.

Return Values

Returns the resulting array.

Examples

Example #30 - array_merge() example

```
<?php
$array1 = array("color" => "red", 2, 4);
$array2 = array("a", "b", "color" => "green", "shape" => "trapezoid", 4);
$result = array_merge($array1, $array2);
print_r($result);
?>
```

The above example will output:

```
Array
(
   [color] => green
```

```
   [0] => 2
   [1] => 4
   [2] => a
   [3] => b
   [shape] => trapezoid
   [4] => 4
)
```

Example #31 - Simple array_merge() example

```
<?php
$array1 = array();
$array2 = array(1 => "data");
$result = array_merge($array1, $array2);
?>
```

Don't forget that numeric keys will be renumbered!

```
Array
(
   [0] => data
)
```

If you want to completely preserve the arrays and just want to append them to each other (not overwriting the previous keys), use the + operator:

```
<?php
$array1 = array();
$array2 = array(1 => "data");
$result = $array1 + $array2;
?>
```

The numeric key will be preserved and thus the association remains.

```
Array
(
   [1] => data
)
```

Warning

The behavior of array_merge() was modified in PHP 5. Unlike PHP 4, array_merge() now only accepts parameters of type array. However, you can use typecasting to merge other types. See the example below for details.

Example #32 - array_merge() PHP 5 example

```
<?php
$beginning = 'foo';
$end = array(1 => 'bar');
$result = array_merge((array)$beginning, (array)$end);
print_r($result);
?>
```

The above example will output:

```
Array
(
   [0] => foo
   [1] => bar
)
```

See Also

- array_merge_recursive()
- array_combine()
- array operators

array_multisort

array_multisort -- Sort multiple or multi-dimensional arrays

Description

bool **array_multisort** (array `$arr` [, mixed `$arg` [, mixed `$...`]])

array_multisort() can be used to sort several arrays at once, or a multi-dimensional array by one or more dimensions.

Associative (string) keys will be maintained, but numeric keys will be re-indexed.

Parameters

arr
An array being sorted.

arg
Optionally another array, or sort options for the previous array argument: **SORT_ASC**, **SORT_DESC**, **SORT_REGULAR**, **SORT_NUMERIC**, **SORT_STRING**.

...
Additional *arg* 's.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #33 - Sorting multiple arrays

```
<?php
$ar1 = array(10, 100, 100, 0);
$ar2 = array(1, 3, 2, 4);
array_multisort($ar1, $ar2);

var_dump($ar1);
var_dump($ar2);
?>
```

In this example, after sorting, the first array will contain 0, 10, 100, 100. The second array will contain 4, 1, 2, 3. The entries in the second array corresponding to the identical entries in the first array (100 and 100) were sorted as well.

```
array(4) {
 [0]=> int(0)
 [1]=> int(10)
 [2]=> int(100)
 [3]=> int(100)
}
array(4) {
 [0]=> int(4)
 [1]=> int(1)
 [2]=> int(2)
 [3]=> int(3)
}
```

Example #34 - Sorting multi-dimensional array

```
<?php
$ar = array(
      array("10", 11, 100, 100, "a"),
      array(   1,  2, "2",   3,   1)
     );
array_multisort($ar[0], SORT_ASC, SORT_STRING,
                $ar[1], SORT_NUMERIC, SORT_DESC);
var_dump($ar);
?>
```

In this example, after sorting, the first array will transform to "10", 100, 100, 11, "a" (it was sorted as strings in ascending order). The second will contain 1, 3, "2", 2, 1 (sorted as numbers, in descending order).

```
array(2) {
 [0]=> array(5) {
   [0]=> string(2) "10"
   [1]=> int(100)
   [2]=> int(100)
   [3]=> int(11)
   [4]=> string(1) "a"
 }
 [1]=> array(5) {
   [0]=> int(1)
   [1]=> int(3)
   [2]=> string(1) "2"
   [3]=> int(2)
   [4]=> int(1)
 }
}
```

Example #35 - Sorting database results

For this example, each element in the *data* array represents one row in a table. This

type of dataset is typical of database records.

Example data:

```
volume | edition
-------+--------
   67 |       2
   86 |       1
   85 |       6
   98 |       2
   86 |       6
   67 |       7
```

The data as an array, called *data*. This would usually, for example, be obtained by looping with mysql_fetch_assoc().

```
<?php
$data[] = array('volume' => 67, 'edition' => 2);
$data[] = array('volume' => 86, 'edition' => 1);
$data[] = array('volume' => 85, 'edition' => 6);
$data[] = array('volume' => 98, 'edition' => 2);
$data[] = array('volume' => 86, 'edition' => 6);
$data[] = array('volume' => 67, 'edition' => 7);
?>
```

In this example, we will order by *volume* descending, *edition* ascending.

We have an array of rows, but array_multisort() requires an array of columns, so we use the below code to obtain the columns, then perform the sorting.

```
<?php
// Obtain a list of columns
foreach ($data as $key => $row) {
   $volume[$key]  = $row['volume'];
   $edition[$key] = $row['edition'];
}

// Sort the data with volume descending, edition ascending
// Add $data as the last parameter, to sort by the common key
array_multisort($volume, SORT_DESC, $edition, SORT_ASC, $data);
?>
```

The dataset is now sorted, and will look like this:

```
volume | edition
-------+--------
   98 |       2
   86 |       1
   86 |       6
   85 |       6
   67 |       2
   67 |       7
```

Example #36 - Case insensitive sorting

Both **SORT_STRING** and **SORT_REGULAR** are case sensitive, strings starting with a capital letter will come before strings starting with a lowercase letter.

To perform a case insensitive search, force the sorting order to be determined by a lowercase copy of the original array.

```
<?php
$array = array('Alpha', 'atomic', 'Beta', 'bank');
$array_lowercase = array_map('strtolower', $array);

array_multisort($array_lowercase, SORT_ASC, SORT_STRING, $array);

print_r($array);
?>
```

The above example will output:

```
Array
(
   [0] => Alpha
   [1] => atomic
   [2] => bank
   [3] => Beta
)
```

array_pad

array_pad -- Pad array to the specified length with a value

Description

array **array_pad** (array `$input`, int `$pad_size`, mixed `$pad_value`)

array_pad() returns a copy of the *input* padded to size specified by *pad_size* with value *pad_value*. If *pad_size* is positive then the array is padded on the right, if it's negative then on the left. If the absolute value of *pad_size* is less than or equal to the length of the *input* then no padding takes place. It is possible to add most 1048576 elements at a time.

Parameters

input
: Initial array of values to pad.

pad_size
: New size of the array.

pad_value
: Value to pad if *input* is less than *pad_size*.

Return Values

Returns a copy of the *input* padded to size specified by *pad_size* with value *pad_value*. If *pad_size* is positive then the array is padded on the right, if it's negative then on the left. If the absolute value of pad_size is less than or equal to the length of the input then no padding takes place.

Examples

Example #37 - array_pad() example

```
<?php
$input = array(12, 10, 9);

$result = array_pad($input, 5, 0);
// result is array(12, 10, 9, 0, 0)

$result = array_pad($input, -7, -1);
// result is array(-1, -1, -1, -1, 12, 10, 9)

$result = array_pad($input, 2, "noop");
// not padded
```

```
?>
```

See Also

- array_fill()
- range()

array_pop

array_pop -- Pop the element off the end of array

Description

mixed **array_pop** (array `&$array`)

array_pop() pops and returns the last value of the *array*, shortening the *array* by one element. If *array* is empty (or is not an array), **NULL** will be returned. Will additionally produce a Warning when called on a non-array.

Note
This function will reset() the array pointer after use.

Parameters

array
The array to get the value from.

Return Values

Returns the last value of *array*. If *array* is empty (or is not an array), **NULL** will be returned.

Examples

Example #38 - array_pop() example

```
<?php
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_pop($stack);
print_r($stack);
?>
```

After this, *$stack* will have only 3 elements:

```
Array
(
    [0] => orange
    [1] => banana
    [2] => apple
)
```

and *raspberry* will be assigned to *$fruit*.

See Also

See also array_push(), array_shift(), and array_unshift().

array_product

array_product -- Calculate the product of values in an array

Description

number **array_product** (array `$array`)

array_product() returns the product of values in an array.

Parameters

`array`
The array.

Return Values

Returns the product as an integer or float.

Examples

Example #39 - array_product() examples

```
<?php

$a = array(2, 4, 6, 8);
echo "product(a) = " . array_product($a) . "\n";

?>
```

The above example will output:

```
product(a) = 384
```

array_push

array_push -- Push one or more elements onto the end of array

Description

int **array_push** (array `&$array`, mixed `$var` [, mixed `$...`])

array_push() treats *`array`* as a stack, and pushes the passed variables onto the end of *`array`*. The length of *`array`* increases by the number of variables pushed. Has the same effect as:

```
<?php
$array[] = $var;
?>
```

repeated for each *`var`*.

Note
If you use array_push() to add one element to the array it's better to use *$array[]* = because in that way there is no overhead of calling a function.

Note
array_push() will raise a warning if the first argument is not an array. This differs from the *$var[]* behaviour where a new array is created.

Parameters

`array`
The input array.

`var`
The pushed value.

Return Values

Returns the new number of elements in the array.

Examples

Example #40 - array_push() example

```
<?php
$stack = array("orange", "banana");
array_push($stack, "apple", "raspberry");
print_r($stack);
?>
```

The above example will output:

```
Array
(
    [0] => orange
    [1] => banana
    [2] => apple
    [3] => raspberry
)
```

See Also

- array_pop()
- array_shift()
- array_unshift()

array_rand

array_rand -- Pick one or more random entries out of an array

Description

mixed **array_rand** (array `$input` [, int `$num_req`])

array_rand() is rather useful when you want to pick one or more random entries out of an array.

Parameters

`input`
The input array.

`num_req`
Specifies how many entries you want to pick - if not specified, defaults to 1.

Return Values

If you are picking only one entry, array_rand() returns the key for a random entry. Otherwise, it returns an array of keys for the random entries. This is done so that you can pick random keys as well as values out of the array.

Examples

Example #41 - array_rand() example

```
<?php
srand((float) microtime() * 10000000);
$input = array("Neo", "Morpheus", "Trinity", "Cypher", "Tank");
$rand_keys = array_rand($input, 2);
echo $input[$rand_keys[0]] . "\n";
echo $input[$rand_keys[1]] . "\n";
?>
```

Notes

Note

As of PHP 4.2.0, there is no need to seed the random number generator with srand()

or mt_srand() as this is now done automatically.

See Also

- shuffle()

array_reduce

array_reduce -- Iteratively reduce the array to a single value using a callback function

Description

mixed **array_reduce** (array `$input`, callback `$function` [, int `$initial`])

array_reduce() applies iteratively the *`function`* function to the elements of the array *`input`* , so as to reduce the array to a single value.

Parameters

`input`
: The input array.

`function`
: The callback function.

`initial`
: If the optional *`initial`* is available, it will be used at the beginning of the process, or as a final result in case the array is empty.

Return Values

Returns the resulting value.

If the array is empty and *`initial`* is not passed, array_reduce() returns **NULL**.

Examples

Example #42 - array_reduce() example

```
<?php
function rsum($v, $w)
{
   $v += $w;
   return $v;
}

function rmul($v, $w)
{
   $v *= $w;
   return $v;
}
```

```
$a = array(1, 2, 3, 4, 5);
$x = array();
$b = array_reduce($a, "rsum");
$c = array_reduce($a, "rmul", 10);
$d = array_reduce($x, "rsum", 1);
?>
```

This will result in *$b* containing *15*, *$c* containing *1200* (= 10*1*2*3*4*5), and *$d* containing *1*.

See Also

- array_filter()
- array_map()
- array_unique()
- array_count_values()

array_reverse

array_reverse -- Return an array with elements in reverse order

Description

array **array_reverse** (array `$array` [, bool `$preserve_keys`])

Takes an input *`array`* and returns a new array with the order of the elements reversed.

Parameters

`array`
The input array.

`preserve_keys`
If set to **TRUE** keys are preserved.

Return Values

Returns the reversed array.

ChangeLog

Version	Description
4.0.3	The *`preserve_keys`* parameter was added.

Examples

Example #43 - array_reverse() example

```
<?php
$input  = array("php", 4.0, array("green", "red"));
$result = array_reverse($input);
$result_keyed = array_reverse($input, true);
?>
```

This makes both *$result* and *$result_keyed* have the same elements, but note the difference between the keys. The printout of *$result* and *$result_keyed* will be:

```
Array
(
    [0] => Array
        (
            [0] => green
            [1] => red
        )

    [1] => 4
    [2] => php
)
Array
(
    [2] => Array
        (
            [0] => green
            [1] => red
        )

    [1] => 4
    [0] => php
)
```

See Also

- array_flip()

array_search

array_search -- Searches the array for a given value and returns the corresponding key if successful

Description

mixed **array_search** (mixed `$needle`, array `$haystack` [, bool `$strict`])

Searches *`haystack`* for *`needle`*.

Parameters

`needle`
The searched value.

Note
If *`needle`* is a string, the comparison is done in a case-sensitive manner.

`haystack`
The array.

`strict`
If the third parameter *`strict`* is set to **TRUE** then the array_search() function will also check the types of the *`needle`* in the *`haystack`*.

Return Values

Returns the key for *`needle`* if it is found in the array, **FALSE** otherwise.

If *`needle`* is found in *`haystack`* more than once, the first matching key is returned. To return the keys for all matching values, use array_keys() with the optional *`search_value`* parameter instead.

Warning
This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

ChangeLog

Version	Description
4.2.0	Prior to PHP 4.2.0, array_search() returns **NULL** on failure instead of **FALSE**.

Examples

Example #44 - array_search() example

```
<?php
$array = array(0 => 'blue', 1 => 'red', 2 => 'green', 3 => 'red');

$key = array_search('green', $array); // $key = 2;
$key = array_search('red', $array);   // $key = 1;
?>
```

See Also

- array_keys()
- array_values()
- array_key_exists()
- in_array()

array_shift

array_shift -- Shift an element off the beginning of array

Description

mixed **array_shift** (array `&$array`)

array_shift() shifts the first value of the *array* off and returns it, shortening the *array* by one element and moving everything down. All numerical array keys will be modified to start counting from zero while literal keys won't be touched.

Note
This function will reset() the array pointer after use.

Parameters

array
The input array.

Return Values

Returns the shifted value, or **NULL** if *array* is empty or is not an array.

Examples

Example #45 - array_shift() example

```
<?php
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_shift($stack);
print_r($stack);
?>
```

The above example will output:

```
Array
(
    [0] => banana
    [1] => apple
    [2] => raspberry
)
```

and *orange* will be assigned to *$fruit*.

See Also

- array_unshift()
- array_push()
- array_pop()

array_slice

array_slice -- Extract a slice of the array

Description

array **array_slice** (array `$array`, int `$offset` [, int `$length` [, bool `$preserve_keys`]])

array_slice() returns the sequence of elements from the array *`array`* as specified by the *`offset`* and *`length`* parameters.

Parameters

`array`
The input array.

`offset`
If *`offset`* is non-negative, the sequence will start at that offset in the *`array`*. If *`offset`* is negative, the sequence will start that far from the end of the *`array`*.

`length`
If *`length`* is given and is positive, then the sequence will have that many elements in it. If *`length`* is given and is negative then the sequence will stop that many elements from the end of the array. If it is omitted, then the sequence will have everything from *`offset`* up until the end of the *`array`*.

`preserve_keys`
Note that array_slice() will reorder and reset the array indices by default. You can change this behaviour by setting *`preserve_keys`* to **TRUE**.

Return Values

Returns the slice.

ChangeLog

Version	Description
5.0.2	The optional *`preserve_keys`* parameter was added.

Examples

Example #46 - array_slice() examples

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

$output = array_slice($input, 2);      // returns "c", "d", and "e"
$output = array_slice($input, -2, 1);  // returns "d"
$output = array_slice($input, 0, 3);   // returns "a", "b", and "c"

// note the differences in the array keys
print_r(array_slice($input, 2, -1));
print_r(array_slice($input, 2, -1, true));
?>
```

The above example will output:

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

See Also

- array_splice()
- unset()

array_splice

array_splice -- Remove a portion of the array and replace it with something else

Description

array **array_splice** (array `&$input`, int `$offset` [, int `$length` [, mixed `$replacement`]])

Removes the elements designated by *`offset`* and *`length`* from the *`input`* array, and replaces them with the elements of the *`replacement`* array, if supplied.

Note that numeric keys in *`input`* are not preserved.

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`*.

Parameters

`input`
: The input array.

`offset`
: If *`offset`* is positive then the start of removed portion is at that offset from the beginning of the *`input`* array. If *`offset`* is negative then it starts that far from the end of the *`input`* array.

`length`
: If *`length`* is omitted, removes everything from *`offset`* to the end of the array. If *`length`* is specified and is positive, then that many elements will be removed. If *`length`* is specified and is negative then the end of the removed portion will be that many elements from the end of the array. Tip: to remove everything from *`offset`* to the end of the array when *`replacement`* is also specified, use *count($input)* for *`length`*.

`replacement`
: If *`replacement`* array is specified, then the removed elements are replaced with elements from this array. If *`offset`* and *`length`* are such that nothing is removed, then the elements from the *`replacement`* array are inserted in the place specified by the *`offset`*. Note that keys in replacement array are not preserved. If *`replacement`* is just one element it is not necessary to put *array()* around it, unless the element is an array itself.

Return Values

Returns the array consisting of the extracted elements.

Examples

Example #47 - array_splice() examples

```
<?php
$input = array("red", "green", "blue", "yellow");
array_splice($input, 2);
// $input is now array("red", "green")

$input = array("red", "green", "blue", "yellow");
array_splice($input, 1, -1);
// $input is now array("red", "yellow")

$input = array("red", "green", "blue", "yellow");
array_splice($input, 1, count($input), "orange");
// $input is now array("red", "orange")

$input = array("red", "green", "blue", "yellow");
array_splice($input, -1, 1, array("black", "maroon"));
// $input is now array("red", "green",
//          "blue", "black", "maroon")

$input = array("red", "green", "blue", "yellow");
array_splice($input, 3, 0, "purple");
// $input is now array("red", "green",
//          "blue", "purple", "yellow");
?>
```

Example #48 - array_splice() examples

The following statements change the values of *$input* the same way:

```
<?php
array_push($input, $x, $y);
array_splice($input, count($input), 0, array($x, $y));
array_pop($input);
array_splice($input, -1);
array_shift($input);
array_splice($input, 0, 1);
array_unshift($input, $x, $y);
array_splice($input, 0, 0, array($x, $y));
$input[$x] = $y; // for arrays where key equals offset
array_splice($input, $x, 1, $y);
?>
```

See Also

- array_slice()
- unset()
- array_merge()

array_sum

array_sum -- Calculate the sum of values in an array

Description

number **array_sum** (array `$array`)

array_sum() returns the sum of values in an array.

Parameters

`array`
The input array.

Return Values

Returns the sum of values as an integer or float.

ChangeLog

Version	Description
4.2.1	PHP versions prior to 4.2.1 modified the passed array itself and converted strings to numbers (which most of the time converted them to zero, depending on their value).

Examples

Example #49 - array_sum() examples

```
<?php
$a = array(2, 4, 6, 8);
echo "sum(a) = " . array_sum($a) . "\n";

$b = array("a" => 1.2, "b" => 2.3, "c" => 3.4);
echo "sum(b) = " . array_sum($b) . "\n";
?>
```

The above example will output:

```
sum(a) = 20
sum(b) = 6.9
```

array_udiff_assoc

array_udiff_assoc -- Computes the difference of arrays with additional index check, compares data by a callback function

Description

array **array_udiff_assoc** (array `$array1`, array `$array2` [, array `$...`], callback `$data_compare_func`)

Computes the difference of arrays with additional index check, compares data by a callback function.

Note
Please note that this function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using, for example, *array_udiff_assoc($array1[0], $array2[0], "some_comparison_func");*.

Parameters

`array1`
The first array.

`array2`
The second array.

`data_compare_func`
The callback comparison function. The user supplied callback function is used for comparison. It must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Return Values

array_udiff_assoc() returns an array containing all the values from *`array1`* that are not present in any of the other arguments. Note that the keys are used in the comparison unlike array_diff() and array_udiff(). The comparison of arrays' data is performed by using an user-supplied callback. In this aspect the behaviour is opposite to the behaviour of array_diff_assoc() which uses internal function for comparison.

Examples

Example #50 - array_udiff_assoc() example

```
<?php
class cr {
   private $priv_member;
   function cr($val)
   {
       $this->priv_member = $val;
   }

   static function comp_func_cr($a, $b)
   {
       if ($a->priv_member === $b->priv_member) return 0;
       return ($a->priv_member > $b->priv_member)? 1:-1;
   }
}

$a = array("0.1" => new cr(9), "0.5" => new cr(12), 0 => new cr(23), 1=> new
cr(4), 2 => new cr(-15),);
$b = array("0.2" => new cr(9), "0.5" => new cr(22), 0 => new cr(3), 1=> new
cr(4), 2 => new cr(-15),);

$result = array_udiff_assoc($a, $b, array("cr", "comp_func_cr"));
print_r($result);
?>
```

The above example will output:

```
Array
(
   [0.1] => cr Object
       (
           [priv_member:private] => 9
       )

   [0.5] => cr Object
       (
           [priv_member:private] => 12
       )

   [0] => cr Object
       (
           [priv_member:private] => 23
       )
)
```

In our example above you see the *"1" => new cr(4)* pair is present in both arrays and thus it is not in the ouput from the function.

See Also

- array_diff()

- array_diff_assoc()
- array_diff_uassoc()
- array_udiff()
- array_udiff_uassoc()
- array_intersect()
- array_intersect_assoc()
- array_uintersect()
- array_uintersect_assoc()
- array_uintersect_uassoc()

array_udiff_uassoc

array_udiff_uassoc -- Computes the difference of arrays with additional index check, compares data and indexes by a callback function

Description

array **array_udiff_uassoc** (array `$array1`, array `$array2` [, array `$...`], callback `$data_compare_func`, callback `$key_compare_func`)

Computes the difference of arrays with additional index check, compares data and indexes by a callback function.

Note that the keys are used in the comparison unlike array_diff() and array_udiff().

Parameters

`array1`
The first array.

`array2`
The second array.

`data_compare_func`
The callback comparison function. The user supplied callback function is used for comparison. It must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. The comparison of arrays' data is performed by using an user-supplied callback : *`data_compare_func`*. In this aspect the behaviour is opposite to the behaviour of array_diff_assoc() which uses internal function for comparison.

`key_compare_func`
The comparison of keys (indices) is done also by the callback function *`key_compare_func`*. This behaviour is unlike what array_udiff_assoc() does, since the latter compares the indices by using an internal function.

Return Values

Returns an array containing all the values from *`array1`* that are not present in any of the other arguments.

Examples

Example #51 - array_udiff_uassoc() example

```
<?php
class cr {
   private $priv_member;
   function cr($val)
   {
       $this->priv_member = $val;
   }

   static function comp_func_cr($a, $b)
   {
       if ($a->priv_member === $b->priv_member) return 0;
       return ($a->priv_member > $b->priv_member)? 1:-1;
   }

   static function comp_func_key($a, $b)
   {
       if ($a === $b) return 0;
       return ($a > $b)? 1:-1;
   }
}
$a = array("0.1" => new cr(9), "0.5" => new cr(12), 0 => new cr(23), 1=> new
cr(4), 2 => new cr(-15),);
$b = array("0.2" => new cr(9), "0.5" => new cr(22), 0 => new cr(3), 1=> new
cr(4), 2 => new cr(-15),);

$result = array_udiff_uassoc($a, $b, array("cr", "comp_func_cr"),
array("cr", "comp_func_key"));
print_r($result);
?>
```

The above example will output:

```
Array
(
   [0.1] => cr Object
       (
           [priv_member:private] => 9
       )

   [0.5] => cr Object
       (
           [priv_member:private] => 12
       )

   [0] => cr Object
       (
           [priv_member:private] => 23
       )
)
```

In our example above you see the *"1" => new cr(4)* pair is present in both arrays and thus it is not in the ouput from the function. Keep in mind that you have to supply 2 callback functions.

Notes

Note
Please note that this function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using, for example, *array_udiff_uassoc($array1[0], $array2[0], "data_compare_func", "key_compare_func");*.

See Also

- array_diff()
- array_diff_assoc()
- array_udiff()
- array_udiff_assoc()
- array_intersect()
- array_intersect_assoc()
- array_uintersect()
- array_uintersect_assoc()
- array_uintersect_uassoc()

array_udiff

array_udiff -- Computes the difference of arrays by using a callback function for data comparison

Description

array **array_udiff** (array `$array1`, array `$array2` [, array `$...`], callback `$data_compare_func`)

Computes the difference of arrays by using a callback function for data comparison. This is unlike array_diff() which uses an internal function for comparing the data.

Parameters

`array1`
The first array.

`array2`
The second array.

`data_compare_func`
The callback comparison function. The user supplied callback function is used for comparison. It must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Return Values

Returns an array containing all the values of *`array1`* that are not present in any of the other arguments.

Examples

Example #52 - array_udiff() example

```
<?php
class cr {
   private $priv_member;
   function cr($val)
   {
       $this->priv_member = $val;
   }

   static function comp_func_cr($a, $b)
   {
```

```
        if ($a->priv_member === $b->priv_member) return 0;
        return ($a->priv_member > $b->priv_member)? 1:-1;
    }
}
$a = array("0.1" => new cr(9), "0.5" => new cr(12), 0 => new cr(23), 1=> new
cr(4), 2 => new cr(-15),);
$b = array("0.2" => new cr(9), "0.5" => new cr(22), 0 => new cr(3), 1=> new
cr(4), 2 => new cr(-15),);

$result = array_udiff($a, $b, array("cr", "comp_func_cr"));
print_r($result);
?>
```

The above example will output:

```
Array
(
    [0.5] => cr Object
        (
            [priv_member:private] => 12
        )

    [0] => cr Object
        (
            [priv_member:private] => 23
        )

)
```

Notes

Note
Please note that this function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using *array_udiff($array1[0], $array2[0], "data_compare_func");*.

See Also

- array_diff()
- array_diff_assoc()
- array_diff_uassoc()
- array_udiff_assoc()
- array_udiff_uassoc()
- array_intersect()
- array_intersect_assoc()
- array_uintersect()
- array_uintersect_assoc()
- array_uintersect_uassoc()

array_uintersect_assoc

array_uintersect_assoc -- Computes the intersection of arrays with additional index check, compares data by a callback function

Description

array **array_uintersect_assoc** (array `$array1`, array `$array2` [, array `$...`], callback `$data_compare_func`)

Computes the intersection of arrays with additional index check, compares data by a callback function.

Note that the keys are used in the comparison unlike in array_uintersect(). The data is compared by using a callback function.

Parameters

`array1`
The first array.

`array2`
The second array.

`data_compare_func`
For comparison is used the user supplied callback function. It must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Return Values

Returns an array containing all the values of *`array1`* that are present in all the arguments.

Examples

Example #53 - array_uintersect_assoc() example

```
<?php
$array1 = array("a" => "green", "b" => "brown", "c" => "blue", "red");
$array2 = array("a" => "GREEN", "B" => "brown", "yellow", "red");

print_r(array_uintersect_assoc($array1, $array2, "strcasecmp"));
?>
```

The above example will output:

```
Array
(
    [a] => green
)
```

See Also

- array_uintersect()
- array_intersect_assoc()
- array_intersect_uassoc()
- array_uintersect_uassoc()

array_uintersect_uassoc

array_uintersect_uassoc -- Computes the intersection of arrays with additional index check, compares data and indexes by a callback functions

Description

array **array_uintersect_uassoc** (array `$array1`, array `$array2` [, array `$...`], callback `$data_compare_func`, callback `$key_compare_func`)

Computes the intersection of arrays with additional index check, compares data and indexes by a callback functions Note that the keys are used in the comparison unlike in array_uintersect(). Both the data and the indexes are compared by using separate callback functions.

Parameters

array1
The first array.

array2
The second array.

data_compare_func
For comparison is used the user supplied callback function. It must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

key_compare_func
Key comparison callback function.

Return Values

Returns an array containing all the values of *array1* that are present in all the arguments.

Examples

Example #54 - array_uintersect_uassoc() example

```
<?php
$array1 = array("a" => "green", "b" => "brown", "c" => "blue", "red");
$array2 = array("a" => "GREEN", "B" => "brown", "yellow", "red");

print_r(array_uintersect_uassoc($array1, $array2, "strcasecmp",
"strcasecmp"));
?>
```

The above example will output:

```
Array
(
    [a] => green
    [b] => brown
)
```

See Also

- array_uintersect()
- array_intersect_assoc()
- array_intersect_uassoc()
- array_uintersect_assoc()

array_uintersect

array_uintersect -- Computes the intersection of arrays, compares data by a callback function

Description

array **array_uintersect** (array `$array1`, array `$array2` [, array `$...`], callback `$data_compare_func`)

Computes the intersection of arrays, compares data by a callback function.

Parameters

`array1`
The first array.

`array2`
The second array.

`data_compare_func`
The callback comparison function. The user supplied callback function is used for comparison. It must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Return Values

Returns an array containing all the values of *`array1`* that are present in all the arguments.

Examples

Example #55 - array_uintersect() example

```
<?php
$array1 = array("a" => "green", "b" => "brown", "c" => "blue", "red");
$array2 = array("a" => "GREEN", "B" => "brown", "yellow", "red");

print_r(array_uintersect($array1, $array2, "strcasecmp"));
?>
```

The above example will output:

```
Array
(
   [a] => green
```

```
    [b] => brown
    [0] => red
)
```

See Also

- array_intersect()
- array_intersect_assoc()
- array_uintersect_assoc()
- array_uintersect_uassoc()

array_unique

array_unique -- Removes duplicate values from an array

Description

array **array_unique** (array `$array`)

Takes an input *`array`* and returns a new array without duplicate values.

Note that keys are preserved. array_unique() sorts the values treated as string at first, then will keep the first key encountered for every value, and ignore all following keys. It does not mean that the key of the first related value from the unsorted *`array`* will be kept.

Note
Two elements are considered equal if and only if *(string) $elem1 === (string) $elem2*. In words: when the string representation is the same. The first element will be used.

Parameters

`array`
The input array.

Return Values

Returns the filtered array.

Examples

Example #56 - array_unique() example

```
<?php
$input = array("a" => "green", "red", "b" => "green", "blue", "red");
$result = array_unique($input);
print_r($result);
?>
```

The above example will output:

```
Array
(
```

```
    [a] => green
    [0] => red
    [1] => blue
)
```

Example #57 - array_unique() and types

```
<?php
$input = array(4, "4", "3", 4, 3, "3");
$result = array_unique($input);
var_dump($result);
?>
```

The above example will output:

```
array(2) {
 [0] => int(4)
 [2] => string(1) "3"
}
```

array_unshift

array_unshift -- Prepend one or more elements to the beginning of an array

Description

int **array_unshift** (array `&$array`, mixed `$var` [, mixed `$...`])

array_unshift() prepends passed elements to the front of the *array*. Note that the list of elements is prepended as a whole, so that the prepended elements stay in the same order. All numerical array keys will be modified to start counting from zero while literal keys won't be touched.

Parameters

array
The input array.

var
The prepended variable.

Return Values

Returns the new number of elements in the *array*.

Examples

Example #58 - array_unshift() example

```
<?php
$queue = array("orange", "banana");
array_unshift($queue, "apple", "raspberry");
print_r($queue);
?>
```

The above example will output:

```
Array
(
    [0] => apple
    [1] => raspberry
    [2] => orange
    [3] => banana
)
```

See Also

- array_shift()
- array_push()
- array_pop()

array_values

array_values -- Return all the values of an array

Description

array **array_values** (array `$input`)

array_values() returns all the values from the *`input`* array and indexes numerically the array.

Parameters

`input`
The array.

Return Values

Returns an indexed array of values.

Examples

Example #59 - array_values() example

```
<?php
$array = array("size" => "XL", "color" => "gold");
print_r(array_values($array));
?>
```

The above example will output:

```
Array
(
   [0] => XL
   [1] => gold
)
```

See Also

- array_keys()

array_walk_recursive

array_walk_recursive -- Apply a user function recursively to every member of an array

Description

bool **array_walk_recursive** (array `&$input`, callback `$funcname` [, mixed `$userdata`])

Applies the user-defined function *`funcname`* to each element of the *`input`* array. This function will recur into deeper arrays.

Parameters

`input`
The input array.

`funcname`
Typically, *`funcname`* takes on two parameters. The *`input`* parameter's value being the first, and the key/index second.

Note
If *`funcname`* needs to be working with the actual values of the array, specify the first parameter of *`funcname`* as a reference. Then, any changes made to those elements will be made in the original array itself.

`userdata`
If the optional *`userdata`* parameter is supplied, it will be passed as the third parameter to the callback *`funcname`*.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #60 - array_walk_recursive() **example**

```
<?php
$sweet = array('a' => 'apple', 'b' => 'banana');
$fruits = array('sweet' => $sweet, 'sour' => 'lemon');

function test_print($item, $key)
```

```
{
   echo "$key holds $item\n";
}

array_walk_recursive($fruits, 'test_print');
?>
```

The above example will output:

```
a holds apple
b holds banana
sour holds lemon
```

You may notice that the key 'sweet' is never displayed. Any key that holds an array will not be passed to the function.

See Also

- array_walk()
- information about the callback type

array_walk

array_walk -- Apply a user function to every member of an array

Description

bool **array_walk** (array `&$array`, callback `$funcname` [, mixed `$userdata`])

Applies the user-defined function *`funcname`* to each element of the *`array`* array.

array_walk() is not affected by the internal array pointer of *`array`*. array_walk() will walk through the entire array regardless of pointer position.

Parameters

`array`
The input array.

`funcname`
Typically, *`funcname`* takes on two parameters. The *`array`* parameter's value being the first, and the key/index second.

Note
If *`funcname`* needs to be working with the actual values of the array, specify the first parameter of *`funcname`* as a reference. Then, any changes made to those elements will be made in the original array itself.

Users may not change the *`array`* itself from the callback function. e.g. Add/delete elements, unset elements, etc. If the array that array_walk() is applied to is changed, the behavior of this function is undefined, and unpredictable.

`userdata`
If the optional *`userdata`* parameter is supplied, it will be passed as the third parameter to the callback *`funcname`*.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Errors/Exceptions

If function *`funcname`* requires more parameters than given to it, an error of level E_WARNING will be generated each time array_walk() calls *`funcname`*. These warnings may be suppressed by prepending the PHP error operator @ to the array_walk() call, or by using error_reporting().

ChangeLog

Version	Description
4.0.0	Passing the key and *`userdata`* to *`funcname`* was added.

Examples

Example #61 - array_walk() example

```
<?php
$fruits = array("d" => "lemon", "a" => "orange", "b" => "banana", "c" =>
"apple");

function test_alter(&$item1, $key, $prefix)
{
   $item1 = "$prefix: $item1";
}

function test_print($item2, $key)
{
   echo "$key. $item2<br />\n";
}

echo "Before ...:\n";
array_walk($fruits, 'test_print');

array_walk($fruits, 'test_alter', 'fruit');
echo "... and after:\n";

array_walk($fruits, 'test_print');
?>
```

The above example will output:

```
Before ...:
d. lemon
a. orange
b. banana
c. apple
... and after:
d. fruit: lemon
a. fruit: orange
b. fruit: banana
c. fruit: apple
```

See Also

- array_walk_recursive()
- create_function()
- list()
- each()
- call_user_func_array()
- array_map()
- information about the callback type
- foreach

array

array -- Create an array

Description

array **array** ([mixed `$...`])

Creates an array. Read the section on the array type for more information on what an array is.

Parameters

...
: Syntax "index => values", separated by commas, define index and values. index may be of type string or integer. When index is omitted, an integer index is automatically generated, starting at 0. If index is an integer, next generated index will be the biggest integer index + 1. Note that when two identical index are defined, the last overwrite the first. Having a trailing comma after the last defined array entry, while unusual, is a valid syntax.

Return Values

Returns an array of the parameters. The parameters can be given an index with the => operator. Read the section on the array type for more information on what an array is.

Examples

The following example demonstrates how to create a two-dimensional array, how to specify keys for associative arrays, and how to skip-and-continue numeric indices in normal arrays.

Example #62 - array() example

```
<?php
$fruits = array (
   "fruits"  => array("a" => "orange", "b" => "banana", "c" => "apple"),
   "numbers" => array(1, 2, 3, 4, 5, 6),
   "holes"   => array("first", 5 => "second", "third")
);
?>
```

Example #63 - Automatic index with array()

```
<?php
$array = array(1, 1, 1, 1,  1, 8 => 1,  4 => 1, 19, 3 => 13);
print_r($array);
?>
```

The above example will output:

```
Array
(
   [0] => 1
   [1] => 1
   [2] => 1
   [3] => 13
   [4] => 1
   [8] => 1
   [9] => 19
)
```

Note that index '3' is defined twice, and keep its final value of 13. Index 4 is defined after index 8, and next generated index (value 19) is 9, since biggest index was 8.

This example creates a 1-based array.

Example #64 - 1-based index with array()

```
<?php
$firstquarter = array(1 => 'January', 'February', 'March');
print_r($firstquarter);
?>
```

The above example will output:

```
Array
(
   [1] => January
   [2] => February
   [3] => March
)
```

As in Perl, you can access a value from the array inside double quotes. However, with PHP you'll need to enclose your array between curly braces.

Example #65 - Accessing an array inside double quotes

```
<?php

$foo = array('bar' => 'baz');
echo "Hello {$foo['bar']}!"; // Hello baz!

?>
```

Notes

Note
array() is a language construct used to represent literal arrays, and not a regular function.

See Also

- array_pad()
- list()
- count()
- range()
- foreach
- The array type

arsort

arsort -- Sort an array in reverse order and maintain index association

Description

bool **arsort** (array `&$array` [, int `$sort_flags`])

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with.

This is used mainly when sorting associative arrays where the actual element order is significant.

Parameters

`array`
The input array.

`sort_flags`
You may modify the behavior of the sort using the optional parameter *`sort_flags`*, for details see sort().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #66 - arsort() example

```
<?php
$fruits = array("d" => "lemon", "a" => "orange", "b" => "banana", "c" =>
"apple");
arsort($fruits);
foreach ($fruits as $key => $val) {
   echo "$key = $val\n";
}
?>
```

The above example will output:

```
a = orange
d = lemon
b = banana
c = apple
```

The fruits have been sorted in reverse alphabetical order, and the index associated with each element has been maintained.

See Also

- asort()
- rsort()
- ksort()
- sort()

asort

asort -- Sort an array and maintain index association

Description

bool **asort** (array `&$array` [, int `$sort_flags`])

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant.

Parameters

`array`
The input array.

`sort_flags`
You may modify the behavior of the sort using the optional parameter *`sort_flags`*, for details see sort().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #67 - asort() example

```
<?php
$fruits = array("d" => "lemon", "a" => "orange", "b" => "banana", "c" =>
"apple");
asort($fruits);
foreach ($fruits as $key => $val) {
   echo "$key = $val\n";
}
?>
```

The above example will output:

```
c = apple
b = banana
d = lemon
a = orange
```

The fruits have been sorted in alphabetical order, and the index associated with each

element has been maintained.

See Also

- arsort()
- sort()
- ksort()
- rsort()

compact

compact -- Create array containing variables and their values

Description

array **compact** (mixed `$varname` [, mixed `$...`])

Creates an array containing variables and their values.

For each of these, compact() looks for a variable with that name in the current symbol table and adds it to the output array such that the variable name becomes the key and the contents of the variable become the value for that key. In short, it does the opposite of extract().

Any strings that are not set will simply be skipped.

Parameters

varname
: compact() takes a variable number of parameters. Each parameter can be either a string containing the name of the variable, or an array of variable names. The array can contain other arrays of variable names inside it; compact() handles it recursively.

Return Values

Returns the output array with all the variables added to it.

Examples

Example #68 - compact() example

```
<?php
$city  = "San Francisco";
$state = "CA";
$event = "SIGGRAPH";

$location_vars = array("city", "state");

$result = compact("event", "nothing_here", $location_vars);
print_r($result);
?>
```

The above example will output:

```
Array
```

```
(
   [event] => SIGGRAPH
   [city] => San Francisco
   [state] => CA
)
```

Notes

Note
Gotcha Because variable variables may not be used with PHP's Superglobal arrays within functions, the Superglobal arrays may not be passed into compact().

See Also

- extract()

count

count -- Count elements in an array, or properties in an object

Description

int **count** (mixed `$var` [, int `$mode`])

Counts elements in an array, or properties in an object.

For objects, if you have SPL installed, you can hook into count() by implementing interface *Countable*. The interface has exactly one method, count(), which returns the return value for the count() function.

Please see the Array section of the manual for a detailed explanation of how arrays are implemented and used in PHP.

Parameters

`var`
The array.

`mode`
If the optional *`mode`* parameter is set to **COUNT_RECURSIVE** (or 1), count() will recursively count the array. This is particularly useful for counting all the elements of a multidimensional array. The default value for *`mode`* is *0*. count() does not detect infinite recursion.

Return Values

Returns the number of elements in *`var`*, which is typically an array, since anything else will have one element.

If *`var`* is not an array or an object with implemented *Countable* interface, *1* will be returned. There is one exception, if *`var`* is **NULL**, *0* will be returned.

Caution
count() may return 0 for a variable that isn't set, but it may also return 0 for a variable that has been initialized with an empty array. Use isset() to test if a variable is set.

ChangeLog

Version	Description
4.2.0	The optional *mode* parameter was added.

Examples

Example #69 - count() example

```
<?php
$a[0] = 1;
$a[1] = 3;
$a[2] = 5;
$result = count($a);
// $result == 3

$b[0]  = 7;
$b[5]  = 9;
$b[10] = 11;
$result = count($b);
// $result == 3

$result = count(null);
// $result == 0

$result = count(false);
// $result == 1
?>
```

Example #70 - Recursive count() example

```
<?php
$food = array('fruits' => array('orange', 'banana', 'apple'),
              'veggie' => array('carrot', 'collard', 'pea'));

// recursive count
echo count($food, COUNT_RECURSIVE); // output 8

// normal count
echo count($food); // output 2

?>
```

See Also

- is_array()
- isset()
- strlen()

current

current -- Return the current element in an array

Description

mixed **current** (array `&$array`)

Every array has an internal pointer to its "current" element, which is initialized to the first element inserted into the array.

Parameters

`array`
The array.

Return Values

The current() function simply returns the value of the array element that's currently being pointed to by the internal pointer. It does not move the pointer in any way. If the internal pointer points beyond the end of the elements list or the array is empty, current() returns **FALSE**.

Warning
This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

Examples

Example #71 - Example use of current() and friends

```
<?php
$transport = array('foot', 'bike', 'car', 'plane');
$mode = current($transport); // $mode = 'foot';
$mode = next($transport);    // $mode = 'bike';
$mode = current($transport); // $mode = 'bike';
$mode = prev($transport);    // $mode = 'foot';
$mode = end($transport);     // $mode = 'plane';
$mode = current($transport); // $mode = 'plane';

$arr = array();
var_dump(current($arr)); // bool(false)
```

```
$arr = array(array());
var_dump(current($arr)); // array(0) { }
?>
```

Notes

Note
You won't be able to distinguish the end of an array from a boolean **FALSE** element. To properly traverse an array which may contain **FALSE** elements, see the each() function.

See Also

- end()
- key()
- each()
- prev()
- reset()
- next()

each

each -- Return the current key and value pair from an array and advance the array cursor

Description

array **each** (array `&$array`)

Return the current key and value pair from an array and advance the array cursor.

After each() has executed, the array cursor will be left on the next element of the array, or past the last element if it hits the end of the array. You have to use reset() if you want to traverse the array again using each.

Parameters

`array`
The input array.

Return Values

Returns the current key and value pair from the array *`array`*. This pair is returned in a four-element array, with the keys *0*, *1*, *key*, and *value*. Elements *0* and *key* contain the key name of the array element, and *1* and *value* contain the data.

If the internal pointer for the array points past the end of the array contents, each() returns **FALSE**.

Examples

Example #72 - each() examples

```
<?php
$foo = array("bob", "fred", "jussi", "jouni", "egon", "marliese");
$bar = each($foo);
print_r($bar);
?>
```

$bar now contains the following key/value pairs:

```
Array
(
   [1] => bob
   [value] => bob
   [0] => 0
   [key] => 0
```

```
)
```

```
<?php
$foo = array("Robert" => "Bob", "Seppo" => "Sepi");
$bar = each($foo);
print_r($bar);
?>
```

$bar now contains the following key/value pairs:

```
Array
(
   [1] => Bob
   [value] => Bob
   [0] => Robert
   [key] => Robert
)
```

each() is typically used in conjunction with list() to traverse an array, here's an example:

Example #73 - Traversing an array with each()

```
<?php
$fruit = array('a' => 'apple', 'b' => 'banana', 'c' => 'cranberry');

reset($fruit);
while (list($key, $val) = each($fruit)) {
   echo "$key => $val\n";
}
?>
```

The above example will output:

```
a => apple
b => banana
c => cranberry
```

Caution

Because assigning an array to another variable resets the original arrays pointer, our example above would cause an endless loop had we assigned *$fruit* to another variable inside the loop.

See Also

- key()
- list()
- current()
- reset()
- next()
- prev()
- foreach

end

end -- Set the internal pointer of an array to its last element

Description

mixed **end** (array `&$array`)

end() advances *`array`* 's internal pointer to the last element, and returns its value.

Parameters

`array`
The array.

Return Values

Returns the value of the last element.

Examples

Example #74 - end() example

```
<?php

$fruits = array('apple', 'banana', 'cranberry');
echo end($fruits); // cranberry

?>
```

See Also

- current()
- each()
- prev()
- reset()
- next()

extract

extract -- Import variables into the current symbol table from an array

Description

int **extract** (array `$var_array` [, int `$extract_type` [, string `$prefix`]])

Import variables from an array into the current symbol table.

extract() checks each key to see whether it has a valid variable name. It also checks for collisions with existing variables in the symbol table.

Parameters

`var_array`
An associative array. This function treats keys as variable names and values as variable values. For each key/value pair it will create a variable in the current symbol table, subject to *`extract_type`* and *`prefix`* parameters. You must use an associative array, a numerically indexed array will not produce results unless you use **EXTR_PREFIX_ALL** or **EXTR_PREFIX_INVALID**.

`extract_type`
The way invalid/numeric keys and collisions are treated is determined by the *`extract_type`*. It can be one of the following values:

EXTR_OVERWRITE
If there is a collision, overwrite the existing variable.

EXTR_SKIP
If there is a collision, don't overwrite the existing variable.

EXTR_PREFIX_SAME
If there is a collision, prefix the variable name with *`prefix`*.

EXTR_PREFIX_ALL
Prefix all variable names with *`prefix`*.

EXTR_PREFIX_INVALID
Only prefix invalid/numeric variable names with *`prefix`*.

EXTR_IF_EXISTS
Only overwrite the variable if it already exists in the current symbol table, otherwise do nothing. This is useful for defining a list of valid variables and then extracting only those variables you have defined out of $_REQUEST, for example.

EXTR_PREFIX_IF_EXISTS
Only create prefixed variable names if the non-prefixed version of the same variable exists in the current symbol table.

EXTR_REFS
Extracts variables as references. This effectively means that the values of the imported variables are still referencing the values of the *var_array* parameter. You can use this flag on its own or combine it with any other flag by OR'ing the *extract_type*.

If *extract_type* is not specified, it is assumed to be **EXTR_OVERWRITE**.

prefix
Note that *prefix* is only required if *extract_type* is **EXTR_PREFIX_SAME**, **EXTR_PREFIX_ALL**, **EXTR_PREFIX_INVALID** or **EXTR_PREFIX_IF_EXISTS**. If the prefixed result is not a valid variable name, it is not imported into the symbol table. Prefixes are automatically separated from the array key by an underscore character.

Return Values

Returns the number of variables successfully imported into the symbol table.

ChangeLog

Version	Description
4.3.0	**EXTR_REFS** was added.
4.2.0	**EXTR_IF_EXISTS** and **EXTR_PREFIX_IF_EXISTS** were added.
4.0.5	This function now returns the number of variables extracted. **EXTR_PREFIX_INVALID** was added. **EXTR_PREFIX_ALL** includes numeric variables as well.

Examples

Example #75 - extract() example

A possible use for extract() is to import into the symbol table variables contained in an associative array returned by wddx_deserialize().

```
<?php

/* Suppose that $var_array is an array returned from
  wddx_deserialize */

$size = "large";
```

```
$var_array = array("color" => "blue",
                   "size"  => "medium",
                   "shape" => "sphere");
extract($var_array, EXTR_PREFIX_SAME, "wddx");

echo "$color, $size, $shape, $wddx_size\n";

?>
```

The above example will output:

```
blue, large, sphere, medium
```

The *$size* wasn't overwritten, because we specified **EXTR_PREFIX_SAME**, which resulted in *$wddx_size* being created. If **EXTR_SKIP** was specified, then $wddx_size wouldn't even have been created. **EXTR_OVERWRITE** would have caused *$size* to have value "medium", and **EXTR_PREFIX_ALL** would result in new variables being named *$wddx_color*, *$wddx_size*, and *$wddx_shape*.

Notes

Warning

Do not use extract() on untrusted data, like user-input ($_GET, ...). If you do, for example, if you want to run old code that relies on register_globals temporarily, make sure you use one of the non-overwriting *extract_type* values such as **EXTR_SKIP** and be aware that you should extract in the same order that's defined in variables_order within the *php.ini*.

See Also

- compact()

in_array

in_array -- Checks if a value exists in an array

Description

bool **in_array** (mixed `$needle`, array `$haystack` [, bool `$strict`])

Searches *`haystack`* for *`needle`*.

Parameters

`needle`
The searched value.

Note
If *`needle`* is a string, the comparison is done in a case-sensitive manner.

`haystack`
The array.

`strict`
If the third parameter *`strict`* is set to **TRUE** then the in_array() function will also check the types of the *`needle`* in the *`haystack`*.

Return Values

Returns **TRUE** if *`needle`* is found in the array, **FALSE** otherwise.

ChangeLog

Version	Description
4.2.0	*`needle`* may now be an array.

Examples

Example #76 - in_array() example

```
<?php
$os = array("Mac", "NT", "Irix", "Linux");
if (in_array("Irix", $os)) {
   echo "Got Irix";
}
if (in_array("mac", $os)) {
   echo "Got mac";
}
?>
```

The second condition fails because in_array() is case-sensitive, so the program above will display:

```
Got Irix
```

Example #77 - in_array() with strict example

```
<?php
$a = array('1.10', 12.4, 1.13);

if (in_array('12.4', $a, true)) {
   echo "'12.4' found with strict check\n";
}

if (in_array(1.13, $a, true)) {
   echo "1.13 found with strict check\n";
}
?>
```

The above example will output:

```
1.13 found with strict check
```

Example #78 - in_array() with an array as needle

```
<?php
$a = array(array('p', 'h'), array('p', 'r'), 'o');

if (in_array(array('p', 'h'), $a)) {
   echo "'ph' was found\n";
}

if (in_array(array('f', 'i'), $a)) {
   echo "'fi' was found\n";
}
```

```
if (in_array('o', $a)) {
   echo "'o' was found\n";
}
?>
```

The above example will output:

```
'ph' was found
 'o' was found
```

See Also

- array_search()
- isset()
- array_key_exists()

key

key -- Fetch a key from an array

Description

mixed **key** (array `&$array`)

key() returns the index element of the current array position.

Parameters

`array`
The array.

Return Values

Returns the index.

Examples

Example #79 - key() example

```
<?php
$array = array(
   'fruit1' => 'apple',
   'fruit2' => 'orange',
   'fruit3' => 'grape',
   'fruit4' => 'apple',
   'fruit5' => 'apple');

// this cycle echoes all associative array
// key where value equals "apple"
while ($fruit_name = current($array)) {
   if ($fruit_name == 'apple') {
       echo key($array).'<br />';
   }
   next($array);
}
?>
```

The above example will output:

```
fruit1<br />
fruit4<br />
fruit5<br />
```

See Also

- current()
- next()

krsort

krsort -- Sort an array by key in reverse order

Description

bool **krsort** (array `&$array` [, int `$sort_flags`])

Sorts an array by key in reverse order, maintaining key to data correlations. This is useful mainly for associative arrays.

Parameters

`array`
The input array.

`sort_flags`
You may modify the behavior of the sort using the optional parameter *`sort_flags`*, for details see sort().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #80 - krsort() example

```
<?php
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
krsort($fruits);
foreach ($fruits as $key => $val) {
   echo "$key = $val\n";
}
?>
```

The above example will output:

```
d = lemon
c = apple
b = banana
a = orange
```

See Also

- asort()
- arsort()
- ksort()
- sort()
- natsort()
- rsort()

ksort

ksort -- Sort an array by key

Description

bool **ksort** (array `&$array` [, int `$sort_flags`])

Sorts an array by key, maintaining key to data correlations. This is useful mainly for associative arrays.

Parameters

`array`
The input array.

`sort_flags`
You may modify the behavior of the sort using the optional parameter *`sort_flags`*, for details see sort().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

ChangeLog

Version	Description
4.0.0	The optional *`sort_flags`* parameter was added.

Examples

Example #81 - ksort() example

```
<?php
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
ksort($fruits);
foreach ($fruits as $key => $val) {
   echo "$key = $val\n";
}
```

```
?>
```

The above example will output:

```
a = orange
b = banana
c = apple
d = lemon
```

See Also

- asort()
- arsort()
- krsort()
- uksort()
- sort()
- natsort()
- rsort()

list

list -- Assign variables as if they were an array

Description

void **list** (mixed `$varname` [, mixed `$...`])

Like array(), this is not really a function, but a language construct. list() is used to assign a list of variables in one operation.

Parameters

varname
A variable.

Return Values

No value is returned.

Examples

Example #82 - list() examples

```
<?php

$info = array('coffee', 'brown', 'caffeine');

// Listing all the variables
list($drink, $color, $power) = $info;
echo "$drink is $color and $power makes it special.\n";

// Listing some of them
list($drink, , $power) = $info;
echo "$drink has $power.\n";

// Or let's skip to only the third one
list( , , $power) = $info;
echo "I need $power!\n";

// list() doesn't work with strings
list($bar) = "abcde";
var_dump($bar); // NULL
?>
```

Example #83 - An example use of list()

```
<table>
<tr>
 <th>Employee name</th>
 <th>Salary</th>
</tr>

<?php

$result = mysql_query("SELECT id, name, salary FROM employees", $conn);
while (list($id, $name, $salary) = mysql_fetch_row($result)) {
   echo " <tr>\n" .
         "  <td><a href=\"info.php?id=$id\">$name</a></td>\n" .
         "  <td>$salary</td>\n" .
         " </tr>\n";
}

?>

</table>
```

Example #84 - Using nested list()

```
<?php

list($a, list($b, $c)) = array(1, array(2, 3));

var_dump($a, $b, $c);

?>

int(1)
int(2)
int(3)
```

Example #85 - Using list() with array indices

```
<?php

$info = array('coffee', 'brown', 'caffeine');

list($a[0], $a[1], $a[2]) = $info;

var_dump($a);

?>
```

Gives the following output (note the order of the elements compared in which order

they were written in the list() syntax):

```
array(3) {
 [2]=>
 string(8) "caffeine"
 [1]=>
 string(5) "brown"
 [0]=>
 string(6) "coffee"
}
```

Notes

Warning

list() assigns the values starting with the right-most parameter. If you are using plain variables, you don't have to worry about this. But if you are using arrays with indices you usually expect the order of the indices in the array the same you wrote in the list() from left to right; which it isn't. It's assigned in the reverse order.

Note

list() only works on numerical arrays and assumes the numerical indices start at 0.

See Also

- each()
- array()
- extract()

natcasesort

natcasesort -- Sort an array using a case insensitive "natural order" algorithm

Description

bool **natcasesort** (array `&$array`)

natcasesort() is a case insensitive version of natsort().

This function implements a sort algorithm that orders alphanumeric strings in the way a human being would while maintaining key/value associations. This is described as a "natural ordering".

Parameters

array
The input array.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #86 - natcasesort() example

```
<?php
$array1 = $array2 = array('IMG0.png', 'img12.png', 'img10.png', 'img2.png',
'img1.png', 'IMG3.png');

sort($array1);
echo "Standard sorting\n";
print_r($array1);

natcasesort($array2);
echo "\nNatural order sorting (case-insensitive)\n";
print_r($array2);
?>
```

The above example will output:

```
Standard sorting
Array
(
   [0] => IMG0.png
   [1] => IMG3.png
```

```
    [2] => img1.png
    [3] => img10.png
    [4] => img12.png
    [5] => img2.png
)

Natural order sorting (case-insensitive)
Array
(
    [0] => IMG0.png
    [4] => img1.png
    [3] => img2.png
    [5] => IMG3.png
    [2] => img10.png
    [1] => img12.png
)
```

For more information see: Martin Pool's » Natural Order String Comparison page.

See Also

- sort()
- natsort()
- strnatcmp()
- strnatcasecmp()

natsort

natsort -- Sort an array using a "natural order" algorithm

Description

bool **natsort** (array `&$array`)

This function implements a sort algorithm that orders alphanumeric strings in the way a human being would while maintaining key/value associations. This is described as a "natural ordering". An example of the difference between this algorithm and the regular computer string sorting algorithms (used in sort()) can be seen in the example below.

Parameters

array
The input array.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #87 - natsort() example

```
<?php
$array1 = $array2 = array("img12.png", "img10.png", "img2.png", "img1.png");

sort($array1);
echo "Standard sorting\n";
print_r($array1);

natsort($array2);
echo "\nNatural order sorting\n";
print_r($array2);
?>
```

The above example will output:

```
Standard sorting
Array
(
    [0] => img1.png
    [1] => img10.png
    [2] => img12.png
    [3] => img2.png
```

```
)

Natural order sorting
Array
(
    [3] => img1.png
    [2] => img2.png
    [1] => img10.png
    [0] => img12.png
)
```

For more information see: Martin Pool's » Natural Order String Comparison page.

See Also

- natcasesort()
- strnatcmp()
- strnatcasecmp()

next

next -- Advance the internal array pointer of an array

Description

mixed **next** (array `&$array`)

next() behaves like current(), with one difference. It advances the internal array pointer one place forward before returning the element value. That means it returns the next array value and advances the internal array pointer by one.

Parameters

`array`
The array being affected.

Return Values

Returns the array value in the next place that's pointed to by the internal array pointer, or **FALSE** if there are no more elements.

Warning

This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

Examples

Example #88 - Example use of next() and friends

```
<?php
$transport = array('foot', 'bike', 'car', 'plane');
$mode = current($transport); // $mode = 'foot';
$mode = next($transport);    // $mode = 'bike';
$mode = next($transport);    // $mode = 'car';
$mode = prev($transport);    // $mode = 'bike';
$mode = end($transport);     // $mode = 'plane';
?>
```

Notes

Note
You won't be able to distinguish the end of an array from a boolean **FALSE** element. To properly traverse an array which may contain **FALSE** elements, see the each() function.

See Also

- current()
- end()
- prev()
- reset()
- each()

pos

pos -- Alias of current()

Description

This function is an alias of: current()

prev

prev -- Rewind the internal array pointer

Description

mixed **prev** (array `&$array`)

Rewind the internal array pointer.

prev() behaves just like next(), except it rewinds the internal array pointer one place instead of advancing it.

Parameters

array
The input array.

Return Values

Returns the array value in the previous place that's pointed to by the internal array pointer, or **FALSE** if there are no more elements.

Examples

Example #89 - Example use of prev() and friends

```
<?php
$transport = array('foot', 'bike', 'car', 'plane');
$mode = current($transport); // $mode = 'foot';
$mode = next($transport);    // $mode = 'bike';
$mode = next($transport);    // $mode = 'car';
$mode = prev($transport);    // $mode = 'bike';
$mode = end($transport);     // $mode = 'plane';
?>
```

Notes

Warning

This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for

more information. Use the === operator for testing the return value of this function.

Note

You won't be able to distinguish the beginning of an array from a boolean **FALSE** element. To properly traverse an array which may contain **FALSE** elements, see the each() function.

See Also

- current()
- end()
- next()
- reset()
- each()

range

range -- Create an array containing a range of elements

Description

array **range** (mixed `$low`, mixed `$high` [, number `$step`])

Create an array containing a range of elements.

Parameters

`low`
: Low value.

`high`
: High value.

`step`
: If a *`step`* value is given, it will be used as the increment between elements in the sequence. *`step`* should be given as a positive number. If not specified, *`step`* will default to 1.

Return Values

Returns an array of elements from *`low`* to *`high`*, inclusive. If low > high, the sequence will be from high to low.

ChangeLog

Version	Description
5.0.0	The optional *`step`* parameter was added.
4.1.0 to 4.3.2	In PHP versions 4.1.0 through 4.3.2, range() sees numeric strings as strings and not integers. Instead, they will be used for character sequences. For example, *"4242"* is treated as *"4"*.
4.1.0	Prior to PHP 4.1.0, range() only generated incrementing integer arrays. Support for character sequences and decrementing arrays was added in 4.1.0. Character

	sequence values are limited to a length of one. If a length greater than one is entered, only the first character is used.

Examples

Example #90 - range() examples

```
<?php
// array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
foreach (range(0, 12) as $number) {
   echo $number;
}

// The step parameter was introduced in 5.0.0
// array(0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
foreach (range(0, 100, 10) as $number) {
   echo $number;
}

// Use of character sequences introduced in 4.1.0
// array('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i');
foreach (range('a', 'i') as $letter) {
   echo $letter;
}
// array('c', 'b', 'a');
foreach (range('c', 'a') as $letter) {
   echo $letter;
}
?>
```

See Also

- shuffle()
- array_fill()
- foreach

reset

reset -- Set the internal pointer of an array to its first element

Description

mixed **reset** (array `&$array`)

reset() rewinds *`array`* 's internal pointer to the first element and returns the value of the first array element.

Parameters

`array`
The input array.

Return Values

Returns the value of the first array element, or **FALSE** if the array is empty.

Examples

Example #91 - reset() example

```
<?php

$array = array('step one', 'step two', 'step three', 'step four');

// by default, the pointer is on the first element
echo current($array) . "<br />\n"; // "step one"

// skip two steps
next($array);
next($array);
echo current($array) . "<br />\n"; // "step three"

// reset pointer, start again on step one
reset($array);
echo current($array) . "<br />\n"; // "step one"

?>
```

See Also

- current()
- each()
- end()
- next()
- prev()

rsort

rsort -- Sort an array in reverse order

Description

bool **rsort** (array `&$array` [, int `$sort_flags`])

This function sorts an array in reverse order (highest to lowest).

Parameters

`array`
The input array.

`sort_flags`
You may modify the behavior of the sort using the optional parameter *`sort_flags`*, for details see sort().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #92 - rsort() example

```
<?php
$fruits = array("lemon", "orange", "banana", "apple");
rsort($fruits);
foreach ($fruits as $key => $val) {
   echo "$key = $val\n";
}
?>
```

The above example will output:

```
0 = orange
1 = lemon
2 = banana
3 = apple
```

The fruits have been sorted in reverse alphabetical order.

Notes

Note
This function assigns new keys to the elements in `array`. It will remove any existing keys that may have been assigned, rather than just reordering the keys.

See Also

- arsort()
- asort()
- ksort()
- krsort()
- sort()
- usort()

shuffle

shuffle -- Shuffle an array

Description

bool **shuffle** (array `&$array`)

This function shuffles (randomizes the order of the elements in) an array.

Parameters

`array`
The array.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #93 - shuffle() example

```
<?php
$numbers = range(1, 20);
srand((float)microtime() * 1000000);
shuffle($numbers);
foreach ($numbers as $number) {
   echo "$number ";
}
?>
```

Notes

Note

This function assigns new keys to the elements in *`array`*. It will remove any existing keys that may have been assigned, rather than just reordering the keys.

Note
As of PHP 4.2.0, there is no need to seed the random number generator with srand() or mt_srand() as this is now done automatically.

See Also

- arsort()
- asort()
- ksort()
- rsort()
- sort()
- usort()

sizeof

sizeof -- Alias of count()

Description

This function is an alias of: count().

sort

sort -- Sort an array

Description

bool **sort** (array `&$array` [, int `$sort_flags`])

This function sorts an array. Elements will be arranged from lowest to highest when this function has completed.

Parameters

`array`
The input array.

`sort_flags`
The optional second parameter *`sort_flags`* may be used to modify the sorting behavior using these values: Sorting type flags:

- **SORT_REGULAR** - compare items normally (don't change types)
- **SORT_NUMERIC** - compare items numerically
- **SORT_STRING** - compare items as strings
- **SORT_LOCALE_STRING** - compare items as strings, based on the current locale. Added in PHP 4.4.0 and 5.0.2. Before PHP 6, it uses the system locale, which can be changed using setlocale(). Since PHP 6, you must use the **i18n_loc_set_default()** function.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

ChangeLog

Version	Description
4.0.0	The *`sort_flags`* parameter was added.

Examples

Example #94 - sort() example

```
<?php

$fruits = array("lemon", "orange", "banana", "apple");
sort($fruits);
foreach ($fruits as $key => $val) {
   echo "fruits[" . $key . "] = " . $val . "\n";
}

?>
```

The above example will output:

```
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange
```

The fruits have been sorted in alphabetical order.

Notes

Note

This function assigns new keys to the elements in *array*. It will remove any existing keys that may have been assigned, rather than just reordering the keys.

Warning

Be careful when sorting arrays with mixed types values because sort() can produce unpredictable results.

See Also

- arsort()
- asort()
- ksort()
- rsort()
- usort()
- uksort()
- array_multisort()
- krsort()
- natsort()

- natcasesort()

uasort

uasort -- Sort an array with a user-defined comparison function and maintain index association

Description

bool **uasort** (array `&$array`, callback `$cmp_function`)

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with, using a user-defined comparison function.

This is used mainly when sorting associative arrays where the actual element order is significant.

Parameters

`array`
The input array.

`cmp_function`
See usort() and uksort() for examples of user-defined comparison functions.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- usort()
- uksort()
- asort()
- arsort()
- ksort()
- rsort()
- sort()

uksort

uksort -- Sort an array by keys using a user-defined comparison function

Description

bool **uksort** (array `&$array`, callback `$cmp_function`)

uksort() will sort the keys of an array using a user-supplied comparison function. If the array you wish to sort needs to be sorted by some non-trivial criteria, you should use this function.

Parameters

`array`
The input array.

`cmp_function`
The callback comparison function. Function *`cmp_function`* should accept two parameters which will be filled by pairs of *`array`* keys. The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #95 - uksort() example

```
<?php
function cmp($a, $b)
{
   $a = ereg_replace('^(a|an|the) ', '', $a);
   $b = ereg_replace('^(a|an|the) ', '', $b);
   return strcasecmp($a, $b);
}

$a = array("John" => 1, "the Earth" => 2, "an apple" => 3, "a banana" => 4);

uksort($a, "cmp");

foreach ($a as $key => $value) {
   echo "$key: $value\n";
}
?>
```

The above example will output:

```
an apple: 3
a banana: 4
the Earth: 2
John: 1
```

See Also

- usort()
- uasort()
- sort()
- asort()
- arsort()
- ksort()
- natsort()
- rsort()

usort

usort -- Sort an array by values using a user-defined comparison function

Description

bool **usort** (array `&$array`, callback `$cmp_function`)

This function will sort an array by its values using a user-supplied comparison function. If the array you wish to sort needs to be sorted by some non-trivial criteria, you should use this function.

Note
If two members compare as equal, their order in the sorted array is undefined.

Note
This function assigns new keys to the elements in *`array`*. It will remove any existing keys that may have been assigned, rather than just reordering the keys.

Parameters

`array`
The input array.

`cmp_function`
The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

ChangeLog

Version	Description
4.1.0	A new sort algorithm was introduced. The *`cmp_function`* doesn't keep the original order

	for elements comparing as equal.

Examples

Example #96 - usort() example

```
<?php
function cmp($a, $b)
{
   if ($a == $b) {
       return 0;
   }
   return ($a < $b) ? -1 : 1;
}

$a = array(3, 2, 5, 6, 1);

usort($a, "cmp");

foreach ($a as $key => $value) {
   echo "$key: $value\n";
}
?>
```

The above example will output:

```
0: 1
1: 2
2: 3
3: 5
4: 6
```

Note

Obviously in this trivial case the sort() function would be more appropriate.

Example #97 - usort() example using multi-dimensional array

```
<?php
function cmp($a, $b)
{
   return strcmp($a["fruit"], $b["fruit"]);
}

$fruits[0]["fruit"] = "lemons";
$fruits[1]["fruit"] = "apples";
```

```
$fruits[2]["fruit"] = "grapes";

usort($fruits, "cmp");

while (list($key, $value) = each($fruits)) {
   echo "\$fruits[$key]: " . $value["fruit"] . "\n";
}
?>
```

When sorting a multi-dimensional array, *$a* and *$b* contain references to the first index of the array.

The above example will output:

```
$fruits[0]: apples
$fruits[1]: grapes
$fruits[2]: lemons
```

Example #98 - usort() **example using a member function of an object**

```
<?php
class TestObj {
   var $name;

   function TestObj($name)
   {
       $this->name = $name;
   }

   /* This is the static comparing function: */
   static function cmp_obj($a, $b)
   {
       $al = strtolower($a->name);
       $bl = strtolower($b->name);
       if ($al == $bl) {
           return 0;
       }
       return ($al > $bl) ? +1 : -1;
   }
}

$a[] = new TestObj("c");
$a[] = new TestObj("b");
$a[] = new TestObj("d");

usort($a, array("TestObj", "cmp_obj"));

foreach ($a as $item) {
   echo $item->name . "\n";
}
?>
```

The above example will output:

```
b
c
d
```

See Also

- uasort()
- uksort()
- sort()
- asort()
- arsort()
- ksort()
- natsort()
- rsort()