Function	Explanation	Example
sizeof($arr)	This function returns the number of elements in an array. Use this function to find out how many elements an array contains; this information is most commonly used to initialize a loop counter when processing the array.	Code: $data = array("red", "green", "blue"); echo "Array has " . sizeof($data) . " elements"; ?> Output: Array has 3 elements
array_values($arr)	This function accepts a PHP array and returns a new array containing only its values (not its keys). Its counterpart is the array_keys() function. Use this function to retrieve all the values from an associative array.	Code: $data = array("hero" => "Holmes", "villain" => "Moriarty"); print_r(array_values($data)); ?> Output: Array ([0] => Holmes [1] => Moriarty)
array_keys($arr)	This function accepts a PHP array and returns a new array containing only its keys (not its values). Its counterpart is the array_values() function. Use this function to	Code: $data = array("hero" => "Holmes", "villain" => "Moriarty"); print_r(array_keys($data)); ?> Output: Array (

	retrieve all the keys from an associative array.	[0] => hero [1] => villain)
array_pop($arr)	This function removes an element from the end of an array.	Code: $data = array("Donald", "Jim", "Tom"); array_pop($data); print_r($data); ?> Output: Array ([0] => Donald [1] => Jim)
array_push($arr, $val)	This function adds an element to the end of an array.	Code: $data = array("Donald", "Jim", "Tom"); array_push($data, "Harry"); print_r($data); ?> Output: Array ([0] => Donald [1] => Jim [2] => Tom [3] => Harry)
array_shift($arr)	This function removes an element from the beginning of an array.	Code: $data = array("Donald", "Jim", "Tom"); array_shift($data); print_r($data);

		?> Output: Array ([0] => Jim [1] => Tom)
array_unshift($arr, $val)	This function adds an element to the beginning of an array.	Code: $data = array("Donald", "Jim", "Tom"); array_unshift($data, "Sarah"); print_r($data); ?> Output: Array ([0] => Sarah [1] => Donald [2] => Jim [3] => Tom)
each($arr)	This function is most often used to iteratively traverse an array. Each time each() is called, it returns the current key-value pair and moves the array cursor forward one element. This makes it most suitable for use in a loop.	Code: $data = array("hero" => "Holmes", "villain" => "Moriarty"); while (list($key, $value) = each($data)) { echo "$key: $value \n"; } ?> Output: hero: Holmes villain: Moriarty

sort($arr)	This function sorts the elements of an array in ascending order. String values will be arranged in ascending alphabetical order. *Note: Other sorting functions include asort(), arsort(), ksort(), krsort() and rsort().*	Code: $data = array("g", "t", "a", "s"); sort($data); print_r($data); ?> Output: Array ([0] => a [1] => g [2] => s [3] => t)
array_flip($arr)	The function exchanges the keys and values of a PHP associative array. Use this function if you have a tabular (rows and columns) structure in an array, and you want to interchange the rows and columns.	Code: $data = array("a" => "apple", "b" => "ball"); print_r(array_flip($data)); ?> Output: Array ([apple] => a [ball] => b)
array_reverse($arr)	The function reverses the order of elements in an array. Use this function to re-order a sorted list of values in reverse for easier processing—for example, when you're trying to begin with the	Code: $data = array(10, 20, 25, 60); print_r(array_reverse($data)); ?> Output: Array ([0] => 60 [1] => 25

	minimum or maximum of a set of ordered values.	[2] => 20 [3] => 10)
array_merge($arr)	This function merges two or more arrays to create a single composite array. Key collisions are resolved in favor of the latest entry. Use this function when you need to combine data from two or more arrays into a single structure—for example, records from two different SQL queries.	Code: $data1 = array("cat", "goat"); $data2 = array("dog", "cow"); print_r(array_merge($data1, $data2)); ?> Output: Array ([0] => cat [1] => goat [2] => dog [3] => cow)
array_rand($arr)	This function selects one or more random elements from an array. Use this function when you need to randomly select from a collection of discrete values—for example, picking a random color from a list.	Code: $data = array("white", "black", "red"); echo "Today's color is " . $data[array_rand($data)]; ?> Output: Today's color is red
array_search($search, $arr)	This function searches the values in an array for a match to the search term, and returns the corresponding key if found. If more than one	Code: $data = array("blue" => "#0000cc", "black" => "#000000", "green" => "#00ff00"); echo "Found " . array_search("#0000cc", $data);

	match exists, the key of the first matching value is returned. Use this function to scan a set of index-value pairs for matches, and return the matching index.	?> Output: Found blue
array_slice($arr, $offset, $length)	This function is useful to extract a subset of the elements of an array, as another array. Extracting begins from array offset $offset and continues until the array slice is $length elements long. Use this function to break a larger array into smaller ones—for example, when segmenting an array by size ("chunking") or type of data.	Code: $data = array("vanilla", "strawberry", "mango", "peaches"); print_r(array_slice($data, 1, 2)); ?> Output: Array ([0] => strawberry [1] => mango)

array_unique($data)	This function strips an array of duplicate values. Use this function when you need to remove non-unique elements from an array—for example, when creating an array to hold values for a table's primary key.	Code: $data = array(1,1,4,6,7,4); print_r(array_unique($data)); ?> Output: Array ([0] => 1 [3] => 6 [4] => 7 [5] => 4)
array_walk($arr, $func)	This function "walks" through an array, applying a user-defined function to every element. It returns the changed array. Use this function if you need to perform custom processing on every element of an array—for example, reducing a number series by 10%.	Code: function reduceBy10(&$val, $key) { $val -= $val * 0.1; } $data = array(10,20,30,40); array_walk($data, 'reduceBy10'); print_r($data); ?> Output: Array ([0] => 9 [1] => 18 [2] => 27 [3] => 36)

- array_change_key_case — Changes the case of all keys in an array
- array_chunk — Split an array into chunks
- array_column — Return the values from a single column in the input array
- array_combine — Creates an array by using one array for keys and another for its values
- array_count_values — Counts all the values of an array
- array_diff_assoc — Computes the difference of arrays with additional index check
- array_diff_key — Computes the difference of arrays using keys for comparison
- array_diff_uassoc — Computes the difference of arrays with additional index check which is performed by a user supplied callback function
- array_diff_ukey — Computes the difference of arrays using a callback function on the keys for comparison
- array_diff — Computes the difference of arrays
- array_fill_keys — Fill an array with values, specifying keys
- array_fill — Fill an array with values
- array_filter — Filters elements of an array using a callback function
- array_flip — Exchanges all keys with their associated values in an array
- array_intersect_assoc — Computes the intersection of arrays with additional index check
- array_intersect_key — Computes the intersection of arrays using keys for comparison
- array_intersect_uassoc — Computes the intersection of arrays with additional index check, compares indexes by a callback function
- array_intersect_ukey — Computes the intersection of arrays using a callback function on the keys for comparison
- array_intersect — Computes the intersection of arrays
- array_key_exists — Checks if the given key or index exists in the array
- array_keys — Return all the keys or a subset of the keys of an array
- array_map — Applies the callback to the elements of the given arrays
- array_merge_recursive — Merge two or more arrays recursively
- array_merge — Merge one or more arrays
- array_multisort — Sort multiple or multi-dimensional arrays
- array_pad — Pad array to the specified length with a value
- array_pop — Pop the element off the end of array
- array_product — Calculate the product of values in an array

- array_push — Push one or more elements onto the end of array
- array_rand — Pick one or more random entries out of an array
- array_reduce — Iteratively reduce the array to a single value using a callback function
- array_replace_recursive — Replaces elements from passed arrays into the first array recursively
- array_replace — Replaces elements from passed arrays into the first array
- array_reverse — Return an array with elements in reverse order
- array_search — Searches the array for a given value and returns the corresponding key if successful
- array_shift — Shift an element off the beginning of array
- array_slice — Extract a slice of the array
- array_splice — Remove a portion of the array and replace it with something else
- array_sum — Calculate the sum of values in an array
- array_udiff_assoc — Computes the difference of arrays with additional index check, compares data by a callback function
- array_udiff_uassoc — Computes the difference of arrays with additional index check, compares data and indexes by a callback function
- array_udiff — Computes the difference of arrays by using a callback function for data comparison
- array_uintersect_assoc — Computes the intersection of arrays with additional index check, compares data by a callback function
- array_uintersect_uassoc — Computes the intersection of arrays with additional index check, compares data and indexes by a callback functions
- array_uintersect — Computes the intersection of arrays, compares data by a callback function
- array_unique — Removes duplicate values from an array
- array_unshift — Prepend one or more elements to the beginning of an array
- array_values — Return all the values of an array
- array_walk_recursive — Apply a user function recursively to every member of an array
- array_walk — Apply a user function to every member of an array
- array — Create an array
- arsort — Sort an array in reverse order and maintain index association
- asort — Sort an array and maintain index association
- compact — Create array containing variables and their values

- count — Count all elements in an array, or something in an object
- current — Return the current element in an array
- each — Return the current key and value pair from an array and advance the array cursor
- end — Set the internal pointer of an array to its last element
- extract — Import variables into the current symbol table from an array
- in_array — Checks if a value exists in an array
- key_exists — Alias of array_key_exists
- key — Fetch a key from an array
- krsort — Sort an array by key in reverse order
- ksort — Sort an array by key
- list — Assign variables as if they were an array
- natcasesort — Sort an array using a case insensitive "natural order" algorithm
- natsort — Sort an array using a "natural order" algorithm
- next — Advance the internal array pointer of an array
- pos — Alias of current
- prev — Rewind the internal array pointer
- range — Create an array containing a range of elements
- reset — Set the internal pointer of an array to its first element
- rsort — Sort an array in reverse order
- shuffle — Shuffle an array
- sizeof — Alias of count
- sort — Sort an array
- uasort — Sort an array with a user-defined comparison function and maintain index association
- uksort — Sort an array by keys using a user-defined comparison function
- usort — Sort an array by values using a user-defined comparison function