Array

H S Rana

CIT,UPES

February 10, 2014

Array

Array is a spacial variable that can hold multiple values at a time.

```
<?php
// define array
$fruits = array('apple', 'banana', 'pineapple', 'grape');
?>
```

Array

Array is a spacial variable that can hold multiple values at a time.

```
<?php
// define array
$fruits = array('apple', 'banana', 'pineapple', 'grape');
?>
```

use index numbers to access the various values stored inside it, with zero representing the first value.
Thus, to access the value 'apple' from the preceding array, you'd use the notation $fruits[0], while the value 'grape' can be retrieved using the notation $fruits[3].

Associative Array

PHP also supports a different type of array, in which index numbers are replaced with user-defined strings, or “keys.”
This type of array is known as an associative array. The keys of the array must be unique; each key references a single value, and the key-value relationship is expressed through the => symbol.

Associative Array

PHP also supports a different type of array, in which index numbers are replaced with user-defined strings, or “keys.”
This type of array is known as an associative array. The keys of the array must be unique; each key references a single value, and the key-value relationship is expressed through the => symbol.

```
<?php
// define array
$fruits = array(
  'a' => 'apple',
  'b' => 'banana',
  'p' => 'pineapple',
  'g' => 'grape'
);
?>
```

Associative Array

PHP also supports a different type of array, in which index numbers are replaced with user-defined strings, or “keys.”
This type of array is known as an associative array. The keys of the array must be unique; each key references a single value, and the key-value relationship is expressed through the => symbol.

```
<?php
// define array
$fruits = array(
  'a' => 'apple',
  'b' => 'banana',
  'p' => 'pineapple',
  'g' => 'grape'
);
?>
```

To access the value 'apple' from the array, you'd use the notation $fruits['a'], while the value 'banana' can be retrieved using the notation $fruits['b']

Assigning Array Values

There are 2 method for assignment

```
1st type
<?php
// define array
$cars = array( 'Ferrari', Porsche', Jaguar',
'Lamborghini', 'Mercedes'
);
?>
2nd type
<?php
// define array
$cars[0] = 'Ferrari';
$cars[1] = 'Porsche';
$cars[2] = 'Jaguar';
$cars[3] = 'Lamborghini';
$cars[4] = 'Mercedes';
?>
```

Assigning Array Values

To automatically assign the next available index to an array value, omit the index number in your array assignment statement.

```
<?php
// define array
$cars[] = 'Ferrari';
$cars[] = 'Lamborghini';
?>
```

Accessing Value

To access a value from an array in a script, simply use the array name and index/key in an expression and PHP will replace it with its value when the script is executed.

```
<?php
// define array
$data = array(
  'username' => 'rahul',
  'password' => 'secret',
  'host' => '192.168.0.1'
);

// use array value
echo 'The password is: ' . $data['password'];
?>
```

Modifying Array Values

simply assign a new value to the element using either its index or its key. To remove an element from an array, use the unset() function on the corresponding key or index:

```
<?php
// define array
$meats = array(
  'fish',
  'chicken',
  'ham',
  'lamb'
);

// remove 'fish'
unset($meats[0]);
?>
```

Modifying Array Values

If you remove an element from an array with unset(), PHP will set that array element to NULL but will not automatically re-index the array. If the array is a numerically indexed array, you can re-index it to remove these “holes” in the indexing sequence, by passing the array through PHP’s array_multisort() function.

Retrieving Array Size

To finding out how many values the array contains, use PHP's count() function, which accepts the array variable as a parameter and returns an integer value.

```
<?php
// define array
$data = array('Monday', 'Tuesday', 'Wednesday');
// get array size
echo 'The array has ' . count($data) . ' elements';
?>
```

Nesting Arrays

PHP also allows you to combine arrays, by placing one inside another to an unlimited depth.

```
<?php
$phonebook = array(
  array(
    'name' => 'Gaurang Kakkar',
    'tel' => '1234567',
    'email' => 'gaurang.kakkar@stu.upes.ac.in',
    ),
  array(
    'name' => 'Shlok Singh',
    'tel' => '8562904',
    'email' => 'shlok_singh@stu.upes.ac.in',
    )
);
echo "Shlok Singh's number is: " . $phonebook[1]['tel'];
?>
```

Processing Arrays with Loops

Consider the following example

```
<?php
// define array
$cities = array('London', 'Paris', 'Madrid', 'Los Angeles',
 'Bombay', 'Jakarta');

// iterate over array
// print each value
for ($i=0; $i<count($cities); $i++) {
  echo $cities[$i] . "\r\n";
}
?>
```

The foreach loop

With a foreach loop, each time the loop runs, the current array element is assigned to a temporary variable, which can then be processed in any way you please—printed, copied to another variable, used in a calculation, and so on. Unlike a for loop, a foreach loop doesn't use a counter; it automatically "knows" where it is in the array, and it moves forward continuously until it reaches the end of the array, at which point it automatically halts.

The foreach loop

With a foreach loop, each time the loop runs, the current array element is assigned to a temporary variable, which can then be processed in any way you please—printed, copied to another variable, used in a calculation, and so on. Unlike a for loop, a foreach loop doesn't use a counter; it automatically "knows" where it is in the array, and it moves forward continuously until it reaches the end of the array, at which point it automatically halts.

```
<?php
// define array
$cities = array('London', 'Paris', 'Madrid', 'Los Angeles',
 'Bombay', 'Jakarta');

// iterate over array
// print each value
foreach ($cities as $c) {
  echo "$c \r\n";
}
```

The foreach loop

The foreach loop also works with associative arrays, except that for such arrays, it uses two temporary variables (one each for the key and value).

```
<?php
// define array
$cities = array(
  "United Kingdom" => "London",
  "United States" => "Washington",
  "France" => "Paris",
  "India" => "Delhi"
);

// iterate over array
// print each value
foreach ($cities as $key => $value) {
  echo "$value is in $key. \r\n";
}
?>
```

explode function

explode function split a string into an array.explode() function, which accepts two arguments—the separator and the source string—and returns an array.

explode function

explode function split a string into an array.explode() function, which accepts two arguments—the separator and the source string—and returns an array.

```
<?php
// define string
$str = 'tinker,tailor,soldier,spy';

// convert string to array
// output: ('tinker', 'tailor', 'soldier, 'spy')
$arr = explode(',', $str);
print_r($arr);
?>
```

implode() Function

Joins array elements into a string.

```
<?php
// define array
$arr = array('one', 'two', 'three', 'four');

// convert array to string
// output: 'one and two and three and four'
$str = implode(' and ', $arr);
print_r($str);
?>
```

Working with Number Range

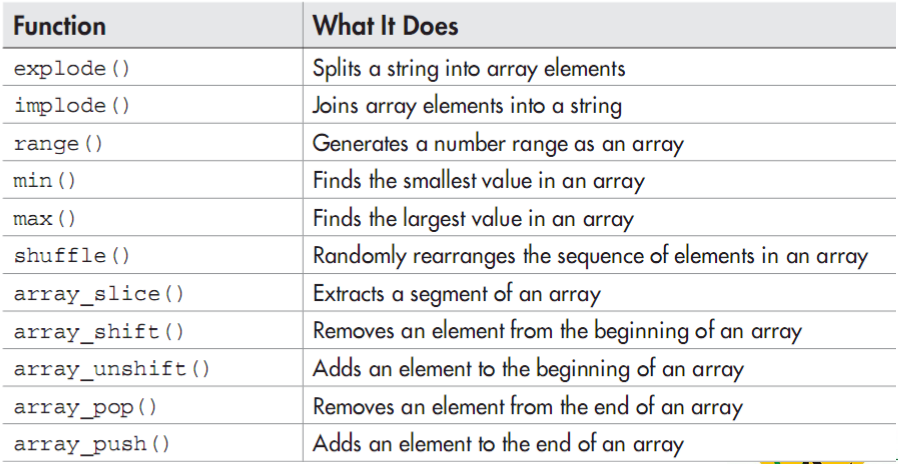
If you're trying to fill an array with a range of numbers, the range() function offers a convenient alternative to manually entering each value. This function accepts two end points and returns an array containing all the numbers between those end points.

Working with Number Range

If you're trying to fill an array with a range of numbers, the range() function offers a convenient alternative to manually entering each value. This function accepts two end points and returns an array containing all the numbers between those end points.

```
<?php
// define array
$arr = range(1,1000);
print_r($arr);
?>
```

Array functions



Function	What It Does
`explode()`	Splits a string into array elements
`implode()`	Joins array elements into a string
`range()`	Generates a number range as an array
`min()`	Finds the smallest value in an array
`max()`	Finds the largest value in an array
`shuffle()`	Randomly rearranges the sequence of elements in an array
`array_slice()`	Extracts a segment of an array
`array_shift()`	Removes an element from the beginning of an array
`array_unshift()`	Adds an element to the beginning of an array
`array_pop()`	Removes an element from the end of an array
`array_push()`	Adds an element to the end of an array

Array's Functions

`array_unique()`	Removes duplicate elements from an array
`array_reverse()`	Reverses the sequence of elements in an array
`array_merge()`	Combines two or more arrays
`array_intersect()`	Calculates the common elements between two or more arrays
`array_diff()`	Calculates the difference between two arrays
`in_array()`	Checks if a particular value exists in an array
`array_key_exists()`	Checks if a particular key exists in an array
`sort()`	Sorts an array
`asort()`	Sorts an associative array by value
`ksort()`	Sorts an associative array by key
`rsort()`	Reverse-sorts an array
`krsort()`	Reverse-sorts an associative array by value
`arsort()`	Reverse-sorts an associative array by key