# Array

- An array in PHP is actually an ordered map. A map is a type that associates *values* to *keys*.

  Syntax to specify array with array() :-

  array( key => value , ... )

  key may only be an integer or string value may be any value of any type

- An array can hold all your variable values under a single name. And you can access the values by referring to the array name and index of element.
- Each element in the array has its own index so that it can be easily accessed.

## Types of Array

There are two kind of arrays:

1. Indexed array - An array with a numeric index beginning with 0. Indexed arrays are used when you identify things by their position.
2. Associative array - An array where each ID key is associated with a value.

   *Associative* arrays have strings as keys and behave more like two-column tables.

   The first column is the key, which is used to access the value.

   PHP internally stores array as associative array.

## Various ways to store data in Indexed array

Assigning elements to array using array():

```
$arrname = array(“val1”,”val2”,”val3”,..);
$cars=array("Saab", "Volvo", "BMW", "Toyota");
```

Manual assignment to initialize array :

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

Adding elements to array at the end:

```
$cars[]="Saab";
$cars[]="Volvo";
$cars[]="BMW";
$cars[]="Toyota";
```

## Various ways to store data in associative array

Using array() with key => value pairs.

```
$arrname = array(“key1”=>val1, “key2”=>val2)

$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Assigning elements at individual index

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

You can specify an initial key with => and then a list of values.

The values are inserted into the array starting with that key, with subsequent values having sequential keys:

```
$days = array(1 => 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday'); // 2 is Tuesday, 3 is Wednesday, etc.
```

# Associative Array

If the initial index is a non-numeric string, subsequent indexes are integers beginning at 0.

```
$whoops = array('Friday' => 'Black', 'Brown', 'Green');
// same as $whoops = array('Friday' => 'Black', 0 => 'Brown', 1 =>
'Green');
```

## Adding values at end of array

To insert more values into the end of an existing indexed array, use the [] syntax:

```
$family = array('Fred', 'Wilma');
$family[] = 'Pebbles'; // $family[2] is 'Pebbles'
```

This construct assumes the array's indexes are numbers and assigns elements into the next available numeric index, starting from 0.

## Assigning a Range of Values

range( ) function creates an array of consecutive integer or character values between the two values you pass to it as arguments.

array range ($start , $limit [, number $step = 1 ] )

**Parameters**

Start - First value of the sequence.

Limit - The sequence is ended upon reaching the *limit* 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.

ArrayDemo.php

```php
<?php
// array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
$number=range(0,12);
		//print_r($number);
foreach($number as $number){
		echo $number." ";
}
echo "<br>";
// The step parameter was introduced in 5.0.0
// array(0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
$numarr=range(0,100,10);
foreach($numarr as $numarr){
		echo $numarr." ";
}
echo "<br>";
// Use of character sequences introduced in 4.1.0
// array('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i');
$letter = range("a","i");
print_r ($letter);
echo "<br>";
// array('c', 'b', 'a');
$letter = range('e','a');
print_r($letter);
?>
```

## Getting size of an Array

**The count( ) and sizeof( ) functions return the number of elements in the array.**

**They are identical in use and effect.**

```
$family = array('Fred', 'Wilma', 'Pebbles');
$size = count($family); // $size is 3
$family = array('Fred', 'Wilma', 'Pebbles');
    $size = sizeof($family); // $size is 3
```

## PADDING AN ARRAY

**To create an array initialized to the same value, use array_pad( ).**

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

*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.

**Example**

```
$scores = array(3, 10);
$padded = array_pad($scores, 5, 0);
// $padded is now array(3, 10, 0, 0, 0)
```

If you want the new values added to the start of the array, use a negative second argument:

```
$padded = array_pad($scores, -5, 0);
```

## EXAMPLE

-
  ```
  <?php
   $scores = array(3, 10);
   $padded = array_pad($scores, 5, 0);
   print_r($padded);
   echo "<br>";
   $padded = array_pad($scores, -5, 0);
   print_r($padded);
   ?>
  ```
  ArrayDemo3.php

## Multidimensional Array

The values in an array can themselves be arrays. This lets you easily create multidimensional arrays:

```
$row_0 = array(1, 2, 3);
$row_1 = array(4, 5, 6);
$row_2 = array(7, 8, 9);
$multi = array($row_0, $row_1, $row_2);
```

You can refer to elements of multidimensional arrays by appending more []s:

```
$value = $multi[2][0];
// row 2, column 0. $value = 7
```

## Extracting Multiple Values

**To copy all of an array's values into variables, use the list( ) construct:**

list(*$variable, ...*) = *$array*;

array's values are copied into the listed variables, in the array's internal order.

Example :-

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
list($n, $a, $w) = $person; // $n is 'Fred', $a is 35, $w is 'Betty'
```

If you have more values in the list( ) than in the array, the extra values are set to NULL:

```
$values = array('hello', 'world');
list($a, $b, $c) = $values;
// $a is 'hello', $b is 'world', $c is NULL
```

## Extracting Multiple Values

If you have more values in the array than in the list( ), the extra values are ignored:

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
list($n, $a) = $person; // $n is 'Fred', $a is 35
```

Two or more consecutive commas in the list( ) skip values in the array:

```
   $values = range('a', 'e');
    list($m,,$n,,$o) = $values; // $m is 'a', $n is 'c', $o is 'e'
```

## Iterator Functions

Every PHP array keeps track of the current element you're working with; the pointer to the current element is known as the *iterator*.

PHP has functions to set, move, and reset this iterator. The iterator functions are:

**current()** : Returns the element currently pointed at by the iterator

**reset( )** : Moves the iterator to the first element in the array and returns it

**next( )** : Moves the iterator to the next element in the array and returns it

## Iterator Functions

**prev():** Moves the iterator to the previous element in the array and returns it

**end() :** Moves the iterator to the last element in the array and returns it

**each():** Returns the key and value of the current element as an array and moves the iterator to the next element in the array key( ) returns the key of the current element.

The each( ) function is used to loop over the elements of an array. It processes elements according to their internal order.

## EXAMPLE

- <?php
```
  $transport = array('foot', 'bike', 'car', 'plane');
  $mode = current($transport);
  print "$mode <br />";
  $mode = next($transport);
  print "$mode <br />";
  $mode = current($transport);
  print "$mode <br />";
  $mode = prev($transport);
  print "$mode <br />";
  $mode = end($transport);
  print "$mode <br />";
  $mode = current($transport);
  print "$mode <br />";
  $mode1 = array();
  $mode1 = each($transport);
  print_r($mode1);
?>
```

ArrayDemo4.php

## Slicing an array

- To extract only a subset of the array, use the array_slice( ) function:

  $subset = array_slice(*array*, *offset*, *length*);
- The array_slice( ) function returns a new array consisting of a consecutive series of values from the original array.
- The *offset* parameter identifies the initial element to copy (0 represents the first element in the array), and the *length* parameter identifies the number of values to copy.
- The new array has consecutive numeric keys starting at 0.

## Slicing an array

- Example

```
$people = array('Tom', 'Dick', 'Harriet', 'Brenda', 'Jo');
$middle = array_slice($people, 2, 2);
// $middle is array('Harriet', 'Brenda')
```

Example:-

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
$subset = array_slice($person, 1, 2);
// $subset is array(0 => 35, 1 => 'Betty')
```

## EXAMPLE

```
<?php
$people = array('Tom', 'Dick', 'Harriet', 'Brenda', 'Jo');
$middle = array_slice($people, 2, 2);
print_r($middle);
echo "<br>";
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
$subset = array_slice($person, 1, 2);
print_r($subset);

?>
```

ArrayDemoSlice.php

## Keys

- The array_keys( ) function returns an array consisting of only the keys in the array, in internal order:
- $array_of_keys = array_keys(*array*);
- Example:

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
$keys = array_keys($person);
// $keys is array('name', 'age', 'wife')
```

## Values

- PHP also provides a function to retrieve an array of just the values in an array, array_values( ):
- $array_of_values = array_values(*array*);
- Example:

  $person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');

  $values = array_values($person); // $values is array('Fred', 35, 'Wilma');

## EXAMPLE

```
<?php
$person = array('name' => 'Fred', 'age' => 35, 'wife' =>
'Wilma');
$keys = array_keys($person);
print_r($keys);
echo "<br>";
$person = array('name' => 'Fred', 'age' => 35, 'wife' =>
'Wilma');
$values = array_values($person);
print_r($values);
?>
```

## Creating variables from array

- The extract( ) function automatically creates local variables from an array. The indexes of the array elements are the variable names

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
extract($person); // $name, $age, and $wife are now set
print $name;
```

- If a variable created by the extraction has the same name as an existing one, the extracted variable overwrites the existing variable.

```
$shape = "round";
$array = array("cover" => "bird", "shape" => "rectangular");
extract($array, EXTR_PREFIX_SAME, "book");
echo "Cover: $cover, Book Shape: $book_shape, Shape: $shape";
```

## EXAMPLE

```
<?php
 $person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
 extract($person); // $name, $age, and $wife are now set
 print $name." ";
 print $age." ";
 print $wife;
 echo "<br>";
 $shape = "round";
 $array = array("cover" => "bird", "shape" => "rectangular");
 extract($array, EXTR_PREFIX_SAME, "book");
 echo "Cover: $cover, Book Shape: $book_shape, Shape: $shape";

 ?>
```

## Creating array from variables

- The compact( ) function is the complement of extract( ).
- Pass it the variable names to compact either as separate parameters or in an array.
- The compact( ) function creates an associative array whose keys are the variable names and whose values are the variable's values.
- Any names in the array that do not correspond to actual variables are skipped.
- $color = 'indigo'; $shape = 'curvy'; $floppy = 'none'; $a = compact('color', 'shape', 'floppy'); // or $names = array('color', 'shape', 'floppy'); $a = compact($names);

## EXAMPLE

```
<?php
 $color="indigo";
 $shape="curvy";
 $floppy="none";
 $temp=compact("color", "shape", "floppy");
 print_r($temp);
 ?>
```

## Searching for elements in array

- The in_array( ) function returns true or false, depending on whether the first argument is an element in the array given as the second argument:

  in_array(*to_find*, *array* )

Example :-

```
<?php
    $addresses = array('spam@cyberpromo.net',
        'abuse@example.com', 'root@example.com');
    $got_spam = in_array('spam@cyberpromo.net', $addresses);
    // $got_spam is true
    echo $got_spam. "<br>";
    $got_milk = in_array('milk@tucows.com', $addresses);
    // $got_milk is false
    echo $got_milk. "<br>";
?>
```

## Sorting one array

| Effect | Ascending | Descending | User-defined order |
|---|---|---|---|
| Sort array by values, then reassign indexes starting with 0 | sort( ) | rsort( ) | usort( ) |
| Sort array by values | asort( ) | arsort( ) | uasort( ) |
| Sort array by keys | ksort( ) | krsort( ) | uksort( ) |

# MERGING ARRAYS

- array_merge( ) function intelligently merges two or more arrays:
  - $merged = array_merge(*array1*, *array2* [, *array* ... ])
  - Indexed Array merge :-

```
$first = array('hello', 'world'); // 0 => 'hello', 1 => 'world'
$second = array('exit', 'here'); // 0 => 'exit', 1 => 'here'
$merged = array_merge($first, $second);
// $merged = array('hello', 'world', 'exit', 'here')
```

  - Associative Array merge :-

```
$first = array('bill' => 'clinton', 'tony' => 'danza');
$second = array('bill' => 'gates', 'adam' => 'west');
$merged = array_merge($first, $second);
// $merged = array('bill' => 'gates', 'tony' => 'danza', 'adam' => 'west')
```

## Printing arrays

- print_r()
  - **print_r()** displays information about a variable in a way that's readable by humans.

  var_dump()
  - This function displays structured information about one or more expressions that includes its type and value.
  - Example :-

```
<?php
  $a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
  print_r ($a);
  ?>
  Array ( [a] => apple [b] => banana [c] => Array ( [0] => x [1] => y [2] =>
  z ) )
```

## CONT…

- Example :-

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

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

## EXPLODE()

- explode() takes a string and splits it into separate elements of an array using the argument provided as a delimiter.
  - explode(separator,string,limit)

| Parameter | Description |
|---|---|
| separator | Required.Delimitor using which you can break string. |
| string | Required. The string to split |
| limit | Optional. Specifies the maximum number of array elements to return. |

  -
    ```
    <?php
    $str = "Hello world. It's a beautiful day.";
    print_r (explode(" ",$str));
    ?>
    ```

# IMPLODE()

- implode() traverses through the elements of an array and re-creates a single string.
- implode(separator, array)

| Parameter | Description |
|---|---|
| separator | Optional. Specifies what to put between the array elements. Default is "" (an empty string) |
| Array | Required. The array to join to a string |

```
<?php
 $arr = array('Hello','World!', 'Beautiful','Day!');
 echo implode(" ",$arr);
 ?>
```

- Output :- Hello World! Beautiful Day!