Unit 2

Functions and Arrays

1. Function

A *function* is a named block of code that performs a specific task, possibly acting upon a set of values given to it, or *parameters*, and possibly returning a single value. The advantages of using functions are:

i. Reducing duplication of code.

ii. Decomposing complex problems into simpler pieces.

iii. Improving clarity of the code.

iv. Reuse of code.

v. Information hiding.

There are two basic types of functions: Built-in functions and user defined functions.

i. **PHP Built-in Functions:** PHP has a huge collection of internal or built-in functions that we can call directly within our PHP scripts to perform a specific task, like gettype(), print_r(), var_dump, etc.

ii. **PHP User-Defined Functions:** In addition to the built-in functions, PHP also allows us to define our own functions. It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately from main program.

Here are some advantages of using functions

a. Functions reduce the repetition of code within a program.

b. Functions make the code much easier to maintain.

c. Functions make it easier to eliminate the errors.

d. Functions can be reused in other application.

2. Calling a Function

Functions in a PHP program can be either built-in (or, by being in an extension, effectively built-in) or user-defined.All functions are evaluated in the same way:

```
$return_value = function_name( [ parameter, ... ] );
```

The number of parameters a function requires differs from function to function. The parameters supplied to the function may be any valid expression and should be in the specific order expected by the function.

Here are some examples of functions

In this *example*, we give an argument, `"vision"`, to the function `strlen()`, which gives us the number of characters in the string it's given.

```
// strlen() is a built-in function that returns the length of a string
$length = strlen("vision"); // $length is now 5
```

The second *example* passes the result of `asin(1)` to the `sin()` function. Since the sine and arcsine functions are reflexive, taking the sine of the arcsine of any value will always return that same value.

```
// sin() and asin() are the sine and arcsine math functions
$result = sin(asin(1)); // $result is the sine of arcsin(1), or 1.0
```

3. Defining a Function

To define a function, use the following syntax

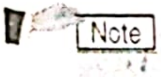
```
function [&] function_name( [ parameter [, ... ] ] )
{
   // Code to be executed
}
```

The declaration of a user-defined function start with the word function, followed by the name of the function we want to create followed by parentheses i.e. () and finally place your function's code between curly brackets {}.

The function name can be any string that starts with a letter or underscore followed by zero or more letters, underscores, and digits. Function names are case-insensitive; that is, we can call the `sin()` function as `sin(1)`, `SIN(1)`, `SiN(1)`, and so on, because all these names refer to the same function.

This is a simple example of an user-defined function

```
<?php
function disp()
{
echo "Welcome to Vision Publications";
}
disp();
?>
//Welcome to Vision Publications
```



A function name must start with a letter or underscore character not with a number, optionally followed by the more letters, numbers, or underscore characters. Function names are case-insensitive.

3.1 PHP Functions Returning Value

A function can return a value using the **return** statement in conjunction with a value or object. Return stops the execution of the function and sends the value back to the calling code.

We can return more than one value from a function using **return array(1, 2, 3, 4)**.

Following *example* takes two integer parameters and multiplies them together and then returns their product to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html>
<head>
   <title>Writing PHP Function which returns value</title>
</head>
<body>
  <?php
     function multiply($num1, $num2)
      {
            $prod = $num1 * $num2;
            return $prod;
         }
         $return_value = multiply(10, 20);
         echo "Returned value from the function : $return_value";
      ?>
</body>
</html>
```

Output
Returned value from the function: 200

3.2 Variable Scope

Functions keep their own sets of variables that are distinct from those of the page and of other functions.

The variables defined in a function, including its parameters, are not accessible outside the function, and, by default, variables defined outside a function are not accessible inside the function. The following *example* illustrates this:

```
$a = 3;
function myfun()
{
  $a += 2;          //outputs '2'
}
myfun();
echo $a;          // outputs '3'
```

The variable $a inside the function myfun() is a different variable than the variable $a outside the variable; even though myfun() uses the add-and-assign operator, the value of the outer $a remains 3 throughout the life of the page. Inside the function, $a has the value 2.

Local Variables

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function –

```
<?php
  $x = 5;
  function assignx()
  {
      $x = 10;
      print "\$x inside function is $x. <br />";
   }
   assignx();
   print "\$x outside of function is $x. <br />";
?>
```

Output
$x inside function is 10.
$x outside of function is 5.

Global Variables

In contrast to local variables, a global variable can be accessed in any part of the program. A global variable must be explicitly declared to be global in the function in which it is to be modified. The keyword GLOBAL has to be written in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name.

Its syntax is

```
global var1, var2, ...
<?php
   $somevar = 15;
   function multi()
   {
      GLOBAL $somevar;
      $somevar=$somevar*$somevar;
      print "Somevar is $somevar"; // Somevar is 225
   }
   multi();
?>
```

Static Variables

Like C, PHP supports declaring function variables *static*. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

We can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

```
static var [= value][, ... ];
```

```
<?php
 function dont_forget()
 {
      STATIC $count = 0;
      $count++;
      print $count;
      print "<br />";
   }
   dont_forget();
   dont_forget();
   dont_forget();
?>
```

Output
1
2
3

4. Function Parameters

Functions can expect, by declaring them in the function definition, an arbitrary number of arguments.

There are two different ways of passing parameters to a function. The first, and more common, is by value. The other is by reference.

4.1 Passing Parameters by Value

In most cases, we pass parameters by value. The argument is any valid expression. That expression is evaluated, and the resulting value is assigned to the appropriate variable in the function.

```
<?php
  function add($num1, $num2)
  {
    $sum = $num1 + $num2;
      echo "Sum of the two numbers is: $sum";
  }
     add(110, 120);
 ?>
// Sum of the two numbers is: 230
```

4.2 Passing Parameters by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. We can pass an argument by reference by adding an ampersand (&) to the variable name in either the function call or the function definition.

Following example depicts both the cases

```
<?php
function nochange($x)
{
    $x=$x*$x+$x;
}
function cube(&$x)
{
     $x = $x * $x * $x;
}
   $result = 5;
   nochange($result);
   echo "call by value to function nochange ".$result;
   cube($result);
   echo "call by reference to function cube ".$result;
?>
```

Output
call by value to function no change 5
call by reference to function cube 125

4.3 Default Parameters

We can set a parameter to have a default value if the function's caller doesn't pass it.

Following function prints NULL in case user does not pass any value to this function.

```
<html>
<head>
    <title>Writing PHP Function which returns value</title>
</head>
<body>
  <?php
   function printMe($param = NULL)
   {
      print $param;
    }
   printMe("This is test");
   printMe();
?>
</body>
</html>
```

Output
This is test

4.4 Variable Parameters

PHP provides three functions we can use in the function to retrieve the parameters passed to it. func_get_args() returns an array of all parameters provided to the function, func_num_args() returns the number of parameters provided to the function, and func_get_arg() returns a specific argument from the parameters.

```
$array = func_get_args();
$count = func_num_args();
$value = func_get_arg(argument_number);
```

Example

```
<?php
function get_arg()
{
    $numargs = func_num_args();
    echo "Number of arguments: $numargs \n";
    if($numargs >= 2)
    {
```

Output
Number of arguments: 2
Second argument is: 22
Argument 0 is: 24
Argument 1 is: 22

```
        echo "Second argument is: " . func_get_arg(1) . "\n";
    }
    $arg_list = func_get_args();
    for($i = 0; $i < $numargs; $i++) {
      echo "Argument $i is: " . $arg_list[$i] . "\n";
    }
}
get_arg(24,22 );
?>
```

4.5 Missing Parameters

PHP allows us when we call a function, we can pass any number of arguments to the function. Any parameters the function expects that are not passed to it remain unset, and a warning is issued for each of them:

```
<?php
function test( $x, $y )
{
  if(isset($x)) { echo " x is set\n"; }
  if(isset($y)) { echo " y is set\n"; }
}
echo "With two arguments:\n";
test(11, 22);
echo "With one argument:\n";
test(100);
?>
```

With two arguments

x is set

y is set

With one argument

PHP Fatal error: Uncaught ArgumentCountError: Too few arguments to function test().

5. Return Values

Values are returned by using the optional return statement. Any type may be returned, including arrays and objects. This causes the function to end its execution immediately and pass control back to the line from which it was called.

```
<?php
function add($num)
{
    return $num + $num;
}
echo add(4);    // outputs '8'.
?>
```

A function cannot return multiple values, but similar results can be obtained by returning an array.

```
<?php
function numbers()
{
    return array (3, 4, 5);
}
list ($zero, $one, $two) =  numbers();
echo $zero.$one.$two; //outputs 3, 4, 5
?>
```

To return a reference from a function, use the reference operator & in both the function declaration and when assigning the returned value to a variable:

```
<?php
function &returns_reference()
{
    return $someref;
}
$newref =&returns_reference();
?>
```

6. Variable Functions

PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it. Among other things, this can be used to implement callbacks, function tables, and so forth.

Variable functions won't work with language constructs such as echo, print, unset(), isset(), empty(), include, require and the like. We need to use your own wrapper function to utilize any of these constructs as variable functions.

```
<?php
  function myfunc($msg)
{
    echo $msg."<br />";
  }
  $func_name = "myfunc";
  $func_name("Hello, This is Vision Publications");
?>
```

This will call the function myfunc() and pass our message "Hello, This is Vision Publications" to it. We accomplish this task by calling the function through the use of the variable $func_name that contains the actual name of the function myfunc.

7. Anonymous Functions

Anonymous functions are similar to regular functions, in that they contain a block of code that is run when they are called. They can also accept arguments, and return values.

The key difference — as their name implies — is that anonymous functions have no name. Here's a code *example* that creates a simple anonymous function:

```
// Declare a basic anonymous function
// (not much use on its own!)
function( $name, $timeOfDay )
{
  return ( "Good $timeOfDay, $name!" );
};
```

1

Apr. 2018 – *4M*
Define Anonymous function. Write anonymous function for addition of two integers.

There are two subtle but important differences between the above example and a regular function definition

i. There is no function name between the function keyword and the opening parenthesis ('('). This tells PHP that we're creating an anonymous function.

ii. There is a semicolon after the function definition. This is because anonymous function definitions are expressions, whereas regular function definitions are code constructs.

Since an anonymous function is an expression much like a number or a string we can do various handy things with it. *For example*, we can:

i. Assign it to a variable, then call it later using the variable's name.

ii. Pass it to another function that can then call it later. This is known as a callback.

iii. Return it from within an outer function so that it can access the outer function's variables. This is known as a closure.

When we define an anonymous function, we can then store it in a variable, just like any other value.

Here's an *example*

```
// Assign an anonymous function to a variable
$makeGreeting = function( $name, $timeOfDay )
{
  return ( "Good $timeOfDay, $name!" );
};
```

Once we've done that, we can call the function using the variable's name, just like we call a regular function:

```
// Call the anonymous function
echo $makeGreeting( "Priyanka", "morning" ) . "<br>";
echo $makeGreeting( "Madhuri", "afternoon" ) . "<br>";
```

Complete *example* is

```
<?php
$makeGreeting= function( $name, $timeOfDay )
{
  return ( "Good $timeOfDay, $name!" );
};
echo $makeGreeting( "Priyanka", "morning" ) . "<br>";
echo $makeGreeting( "Madhuri", "afternoon" ) . "<br>";
?>
```

Output

Good morning, Priyanka!
Good afternoon, Madhuri!

Programs

1. Program to count total number of vowels in a string.

```
<?php
function cnt_vowels($a,$l)
{
$v_cnt = 0;
for($i = 0;$i < $l;$i++)
{
   if(( $a[$i] == 'a') || ($a[$i] == 'e') || ($a[$i] == 'i') || ($a[$i]
==   'o') || ($a[$i] == 'u') || ($a[$i] == 'A') || ($a[$i] == 'E') ||
($a[$i]   == 'I') || ($a[$i] == 'O') || ($a[$i] == 'U'))
$v_cnt++;
 }
 return $v_cnt;
}
 function occur_vowels($a,$l)
  {
    $av=$ev=$iv=$ov=$uv=$cnt=$tot=0;
   for($i=0;$i<$l;$i++)
   {
     if(( $a[$i] == 'a') || ($a[$i] == 'A'))
        $av++;
     else if(( $a[$i] == 'e') || ($a[$i] == 'E'))
        $ev++;
   else if(( $a[$i] == 'i') || ($a[$i] == 'I'))
     $iv++;
   else if(( $a[$i] == 'o') || ($a[$i] == 'O'))
     $ov++;
   else if(( $a[$i] == 'u') || ($a[$i] == 'U'))
     $uv++;
    }

    echo "<br> Total 'a' : $av";
    echo "<br> Total 'e' : $ev";
    echo "<br> Total 'i' : $iv";
    echo "<br> Total 'o' : $ov";
    echo "<br> Total 'u' : $uv\n";
    $tot =  $av+$ev+$iv+$ov+$uv;
    return $tot;
     }
$str="vision publications";
$l = strlen($str);
$c=cnt_vowels($str,$l);
echo "\nTotal vowels in $str are $c\n<br>";
$v_cnt = occur_vowels($str,$l);
echo "<br>Total vowels are $v_cnt";
?>
```

Output

```
Total vowels in vision
publications are 8
Total 'a' : 1
Total 'e' : 0
Total 'i' : 4
Total 'o' : 2
Total 'u' : 1
Total vowels are 8
```

2. Write a PHP script to accept the number from user and write a function to calculate the factorial of a given number (non-negative integer). The function accept the number as argument.

```
<?php
/* Function to get Factorial of a Number */
function Factorial($num)
{
   $fact = 1;
  for($i = 1; $i <= $num ;$i++)
  $fact = $fact * $i;
  return $fact;
}
?>
<!doctype html>
<html>
<head>
<title>Factorial Program using PHP</title>
</head>
<body>
<form action = "" method="post">
  Enter the number whose factorial requires <br />
<input type="number" name="number" />
<input type="submit" name="submit" value="Submit" />
</form>
<?php
if(isset($_POST['submit']) and $_POST['submit'] == "Submit")
{
    if(isset($_POST['number']) and is_numeric($_POST['number']))
    echo 'Factorial Of Number:
  <strong>'.Factorial($_POST['number']).'</strong>';
  }
?>
</body>
</html>
```

3. Write a PHP script for the following:

a. Design a form to accept two numbers from the users.

b. Give option to choose an arithmetic operation (use Radio Button).

c. Display the result on next form.

d. Use concept of default parameter.

Airth.inc

```
<?php
function add($a=0,$b=0)
{
   return $a+$b;
}
function sub($a=0,$b=0)
{
   return $a-$b;
}
function mult($a=0,$b=0)
{
   return $a*$b;
}
function div($a=0,$b=1)
{
   return $a/$b;
}
function mod($a=0,$b=1)
{
   return $a%$b;
}
?>
```

Airth.php

```
<?php
include 'airth.inc';
$x=$_POST['n1'];
$y=$_POST['n2'];
$op=$_POST['op'];
switch($op)
{
case 1:if(isset($x) && isset($y))
       printf("Answer=%d",add($x,$y));
         else
       printf("Answer=%d",add());
         break;
case 2:if(isset($x) && isset($y))
       printf("Answer=%d",sub($x,$y));
         else
       printf("Answer=%d",sub());
         break;
```

```
case 3:if(isset($x) && isset($y))
        printf("Answer=%d",mult($x,$y));
          else
        printf("Answer=%d",mult());
          break;
case 4:if(isset($x) && isset($y))
        printf("Answer=%d",div($x,$y));
          else
        printf("Answer=%d",div());
          break;
case 5:if(isset($x) && isset($y))
        printf("Answer=%d",mod($x,$y));
          else
        printf("Answer=%d",mod());
          break;
}
?>
```

Airth.html

```
<html>
<body>
<form method='post' action='airth.php'>
  Number1:<input type='text' name='n1'><br>
  Number2:<input type='text' name='n2'><br>
  <b>Select option :</b><br>
       <input type='radio' name='op' value=1>add(+)
       <input type='radio' name='op' value=2>sub(-)
       <input type='radio' name='op' value=3>mult(*)
       <input type='radio' name='op' value=4>div(/)
       <input type='radio' name='op' value=5>mod(%)
    <br>
      <input type='submit'>
      <input type='reset'>
</form>
</body>
</html>
```

8. Arrays

Arrays are complex variables that allow us to store more than one value or a group of values under a single variable name. *For example,* if we want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.

9. Indexed vs Associative Arrays

There are two kinds of arrays in PHP: indexed and associative.

Indexed **array** keys are integers, beginning at 0. Indexed arrays are used when we identify things by their position.

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 all arrays as associative arrays, so the only difference between associative and indexed arrays is what the keys happen to be.

Indexed Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

Example

Following is the *example* showing how to create and access numeric arrays. Here we have The **array()** function to create array.

There are two ways to define indexed array

1st way

```
$season=array("summer","winter","spring","autumn");
```

2nd way

```
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative arrays will have their index as string so that we can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

Note Don't keep associative array inside double quote while printing otherwise it would not return any value.

There are two ways to create an associative array

```
$age = array("S"=>"Summer", "W"=>"Winter", "R"=>"Rainy");
```

or

```
$age['S'] = "Summer";
$age['W'] = "Winter";
$age['R'] = "Rainy";
```

10. Identifying Elements of an Array

To access specific values from an array use the array variable's name, followed by the element's key (sometimes called the *index*) within square brackets:

Syntax: `arrayname[index];`

Example

```
$name['vision publications']
$year[2017]
```

1

Apr. 2018 – *1M*
Which function identifies a subset of an array based on its values?

The key can be either a string or an integer. String values that are equivalent to integer numbers (without leading zeros) are treated as integers.

11. Storing Data in Arrays

Storing a value in an array will create the array. We can define an array by directly assigning a value to an element. Assume that **$emp1** array hasn't been defined yet.

```
$emp1[0] = 'Ram' // Makes $emp1 an array
```

For Associative arrays,

```
$emp1['name'] = 'Ram' // Makes $emp1 an array
```

An easier way to initialize an array is to use the `array()` construct, which builds an array from its arguments:

```
$emp = array('Ram', 'shyam', 'ghanshyam');
```

To create an associative array with `array( )`, use the => symbol to separate indexes from values:

```
$emp = array('ram' => 150000,
             'shyam'  => 750000,
             'ghanshyam'   => 50000);
```

To construct an empty array, pass no arguments to `array()`:

```
$emp = array();
```

We 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:

```
$months = array(1 => 'january', 'february', 'march', 'april', 'may',
'june', 'july', 'august', 'september', 'october', 'november', 'december');
// 2 is february, 3 is march, etc.
```

11.1 Adding Values to the End of an Array

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

```
$emp = array('ram', 'shyam');
$emp[] = 'sita';                    // $emp[2] is 'sita'
```

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

11.2 Assigning a Range of Values

The range() function creates an array containing a range of elements.

Syntax: `range(low,high,step)`

low :Specifies the lowest value of the array.

high: Specifies the highest value of the array.

step is optional: Specifies the increment used in the range. Default is 1.

For example

```
$number = range(1, 5);          // $number = array(1,2, 3, 4, 5);
$letter = range('a', 'z');      // $letter holds the alphabet
$reverse= range(5, 1);          // $reverse = array(5, 4, 3, 2,1);
```

Only the first letter of a string argument is used to build the range

```
range('aaa', 'zzz')             /// same as range('a','z')
```

11.3 Getting the Size of an Array

The `count()` and `sizeof()` functions return the number of elements in the array. Here's an *example*:

```
$emp = array('ram', 'shyam', 'ghanshyam');
$size = count($emp);                    // $size is 3
```

11.4 Padding an Array

The `array_pad()` function inserts a specified number of elements, with a specified value, to an array.

Syntax: `array_pad(array, size, value)`

The first argument to `array_pad()` is the array, the second argument is the minimum number of elements you want the array to have, and the third argument is the value to give any elements that are created. The `array_pad()` function returns a new padded array, leaving its argument array alone.

Here's `array_pad()` in action:

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

Notice how the new values are appended to the end of the array. If we want the new values added to the start of the array, use a negative second argument:

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

12. Multidimensional Arrays

PHP multidimensional array is also known as array of arrays. It allows us to store tabular data in an array. PHP multidimensional array can be represented in the form of matrix which is represented by row X column.

1

Apr. 2018 – *4M*
Describe Multidimensional array with example.

```
<html>
<body>
  <?php
    $stud = array(
              "xyz" => array (
              "physics" => 35,
              "maths" => 30,
              "chemistry" => 39
            ),
      "abc" => array (
              "physics" => 30,
              "maths" => 32,
              "chemistry" => 29
            ),
"pqr" => array(
                    "physics" => 31,
                    "maths" => 22,
                    "chemistry" => 39
                )
        );
       /* Accessing multi-dimensional array values */
         echo "Marks for xyz in physics : " ;
         echo $stud['xyz']['physics'] . "<br />";
         echo "Marks for abc in maths : ";
         echo $stud['abc']['maths'] . "<br />";
         echo "Marks for pqr in chemistry : " ;
         echo $stud['pqr']['chemistry'] . "<br />";
      ?>
   </body>
</html>
```

Output
Marks for xyz in physics : 35
Marks for abc in maths : 32
Marks for pqr in chemistry : 39

13. Extracting Multiple Values

Php provides functions to extract multiple values from array.

The list() function is used to assign values to a list of variables in one operation.

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

The array's values are copied into the listed variables, in the array's internal order. By default that's the order in which they were inserted.

Here's an example

```
<?php
   $car= array("wagonR","Alto","Swift");
      list($a, $b, $c) = $car;
   echo "I have several cars, a $a, a $b, and a $c.";
?>
```

Output
I have several cars, a wagonR, a Alto, and a Swift.

If we have more values in the array than in the `list()`, the extra values are ignored

```
$car= array("wagonR","Alto","Swift","Suzuki ertiga","eeco");
list($a,$b)=$car;  // $a is wagonR, $b is Alto
```

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

```
$car= array("wagonR","Alto");
List($a, $b, $c)=$car;  // $a is wagonR, $b is Alto, $c is NULL
```

13.1 Slicing an Array

The function returns the sequence of elements from the array as specified by the offset and length parameters.

```
array_slice($array, $offset [,$length [,$preserve_keys]] );
```

The `array_slice()` function returns a new array consisting of a consecutive series of values from the original array.

The offset parameter specifies where the function will start the slice, and the length specifies the length of the returned array. If this value is set to a negative number, the function will stop

slicing that far from the last element. If this value is not set, the function will return all elements, starting from the position set by the start-parameter.

For example

```
$emp = array('aaa', 'bbb', 'ccc', 'ddd', 'eee');
$new_emp = array_slice($emp, 2, 2);
print_r($new_emp);
```

Output
Array([0] => ccc [1] => ddd)

Example

```
<!DOCTYPE html>
<html>
<body>
  <?php
     $a=array("aaa"=>"red","bbb"=>"green","ccc"=>"blue","ddd"=>"yellow"
       , "eee"=>"brown");
     print_r(array_slice($a,1,2));

$a=array("000"=>"red","111"=>"green","222"=>"blue","333"=>"yellow"
        ,"444"=>"brown");
    print_r(array_slice($a,1,2));
?>
</body>
</html>
```

Output
Array ([bbb] => green [ccc] => blue)
Array ([0] => green [1] => blue)

13.2 Splitting an Array into Chunks

The array_chunk() function splits an array into chunks of new arrays.

```
$chunks = array_chunk(array, size [, preserve_keys]);
```

The function returns an array of the smaller arrays.

The second argument is an integer that specifies the size of each chunk.

The third argument, `preserve_keys`, is a boolean value that determines whether the elements of the new arrays have the same keys as in the original or new numeric keys starting from 0.

```
<!DOCTYPE html>
<html>
<body>
<?php
   $age=array("A"=>"555","B"=>"777","C"=>"444","D"=>"000");
   print_r(array_chunk($age,2,true));
   //Array ( [0] => Array ( [A] => 555 [B] => 777 )
             [1] => Array ( [C] => 444 [D] => 000 ) )
```

```
  print_r(array_chunk($age,3,true));
// Array ( [0] => Array ( [A] => 555 [B] => 777 [C] => 444 )
           [1] => Array ( [D] => 000 ) )
print_r(array_chunk($age,1,true));
// Array ( [0] => Array ( [A] => 555 )
           [1] => Array ( [B] => 777 )
           [2] => Array ( [C] => 444 )
           [3] => Array ( [D] => 000 ) )
?>
</body>
</html>
```

13.3 Keys and Values

`array_keys()` and `array_values()` are very closely related functions: the former returns an array of all the keys in an array, and the latter returns an array of all the values in an array.

`array_keys()` function accepts a PHP array and returns a new array containing only its keys (not its values). Use this function to retrieve all the keys from an associative array.

Syntax: `array_keys( $input [, $search_value [, $strict]] );`

search_value: specifies a value, then only the keys with this value are returned.

Strict is Optional. Used with the value parameter.

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
  $fruits=array("aaa"=>"Apple","bbb"=>"Banana","ccc"=>"Oranges");
  print_r(array_keys($fruits));
  print_r(array_keys($fruits,'Banana'));
?>
</body>
</html>
```

Output
Array ([0] => aaa [1] => bbb [2] => ccc)
Array ([0] => bbb)

`array_values()`: 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.

Syntax: `$array_of_values = array_values(array);`

```
$fruits=array("aaa"=>"Apple","bbb"=>"Banana","ccc"=>"Oranges");
print_r(array_values($fruits));
```

Output
Array([0] => Apple [1] => Banana [2] => Oranges)

13.4 Checking Whether an Element Exists

The `array_key_exists()` function is used to check if an element exists in the array,

```
if(array_key_exists(key, array)) { ... }
```

The function returns TRUE if the given key is set in the array otherwise FALSE.

```
<?php
   $fruit = array('apples' => 100, 'banana' => 400,'oranges'=> 300);
   if(array_key_exists('banana', $fruit))
   {
       echo "The 'fruit' is in the array";
   }
   else
   echo "No such fruit in the array";
?>
```

Output
The 'fruit' is in the array

13.5 Removing and Inserting Elements in an Array

The `array_splice()` function removes the elements designated by **offset** and **length** from the **input** array, and replaces them with the elements of the **replacement** array, if supplied. It returns an array containing the extracted elements.

```
array_splice( $input, $offset [,$length [,$replacement]] );
```

input: Specifies an array.

offset: Specifies where the function will start removing elements. 0 = the first element.

length(Optional): Specifies how many elements will be removed, and also length of the returned array.

replacement(Optional): Specifies an array with the elements that will be inserted to the original array.

```
<?php
   $input = array("apple", "banana", "oranges", "mangoes");
```

```
    array_splice($input, 2);
    print_r($input);
    print_r("<br />");
    $input = array("apple", "banana", "oranges", "mangoes");
    array_splice($input, 1,-2);
    print_r($input);
    print_r("<br />");
    $input = array("apple", "banana", "oranges", "mangoes");
    array_splice($input, 1, count($input), "grapes");
    print_r($input);
    print_r("<br />");
    $input = array("apple", "banana", "oranges", "mangoes");
    array_splice($input,-1, 1,array("banana","grapes"));
    print_r($input);
    print_r("<br />");
  ?>
Array ( [0] => apple [1] => banana )
Array ( [0] => apple [1] => oranges [2] => mangoes )
Array ( [0] => apple [1] => grapes )
Array ( [0] => apple [1] => banana [2] => oranges [3] => banana [4] =>
grapes )
```

14. Traversing Arrays

The most common task with arrays is to do something with every element—for instance, sending mail to each element of an array of addresses, updating each file in an array of filenames, or adding up each element of an array of prices. There are several ways to traverse arrays in PHP, and the one you choose will depend on your data and the task you're performing.

14.1 The foreach Construct

The `foreach` loop statement allows us to iterate over elements of an array or public properties of an object. The `foreach` loop statement only works with arrays and objects. If we use the `foreach` loop statement with other data types, we will get an error.

```
foreach (array as value)
{
  code to be executed;
}
```

```
<?php
    $array = array( 11, 22,33, 44, 55);
    foreach( $array as $value )
{
      echo "Value is $value <br />";
      }
?>
```

Output
Value is 11
Value is 22
Value is 33
Value is 44
Value is 55

To change the values of array elements inside the loop, we have to use a reference that returns by the `foreach` loop. To enable the `foreach` loop to return a reference to the array element, we add an ampersand (&) symbol in front of the loop variable as follows:

```
<?php
foreach ($array as &$value)
{
}
```

The following *example* changes values of elements of an indexed array inside the loop:

```
<?php
$scores = [1,2,3];
foreach ($scores as &$score)
{
 $score *= 2;
}
 print_r($scores); // [2, 4, 6]
```

PHP foreach – iterate over elements of an associative array

To loop over elements of an associative array, use the following syntax

```
<?php
 foreach ($array as $key => $value)
{
   //process element here;
}
```

For each element in the array, the key is assigned to the `$key` variable and the value is assigned to the `$value` variable.

Let's see the following example

```
<?php
$name = [
    'fname'  => 'amar',
    'lname'   => 'upadhyay',
    'mname' => 'jignesh'
```

Output
fname:amar
lname:upadhyay
mname:jignesh

```
];
foreach ($name as $key => $value)
{
    echo $key . ':' . $value . '<br>';
}
```

14.2 The Iterator Functions

PHP array keeps track of the current element we'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.

`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.

```
<html>
   <body>
      <?php
$result = array( 'Subject'  => 'Marks',
                 'Maths' => 35,
               'Hindi' => 30,
             'English' => 45,
             'Science' => 50);
// start table and print heading
reset($result);
list($sub, $marks) = each($result);
echo("<table border=1><tr><th>$sub</th>
    <th>$marks</th></tr>\n");
// print the rest of the values
while (list($sub,$marks) = each($result))
```

Output

Subject	Marks
Maths	35
Hindi	30
English	45
Science	50

```
{
    echo("<tr><td>$sub</td><td>$marks</td></tr>\n");
}
// end the table
echo("</table>");
?>
   </body>
</html>
```

14.3 Using a for Loop

The `for` loop operates on the array itself, not on a copy of the array, and processes elements in key order regardless of their internal order. The for loop is used when we know in advance how many times the script should run.

```
for (initialization; condition; increment)
{
   code to be executed;
}
```

The initializer is used to set the start value for the counter of the number of loop iterations.

Here's how to print an array using ***for***

```
<html>
<body>
  <?php
     $fruit = array('mango', 'apple','banana');
    for($i = 0; $i < count($fruit); $i++)
     {
        $value = $fruit[$i];
        echo "$value\n";
     }
   ?>
</body>
</html>
```

14.4 Calling a Function for Each Array Element

The array_walk() function runs each array element in a user-defined function. The array's keys and values are parameters in the function.

Syntax

```
array_walk(array, function_name, parameter);
```

where array is the name of the array and function_name is the name of the user-defined function. Parameter is optional which specifies that we can assign one parameter to the function, or as many as we like.

```
<?php
function myfun($value,$key)
{
   echo " $key for $value<br />";
}
$fruit=array("A"=>"Apple","B"=>"Banana","C"=>"Chickoo");
array_walk($fruit,"myfun");
?>
```

Output
A for Apple
B for Banana
C for Chickoo

14.5 Reducing an Array

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

```
array_reduce( $array, callback $function [, int $initial] );
```

function: This is a callback function.

initial is optional, which specifies the initial value to send to the function.

```
<?php
function sum($carry, $item)
{
    $carry += $item;
    return $carry;
}
function multiplication($carry, $item)
{   $carry *= $item;
    return $carry;
}
```

```
  $a = array(1, 2, 3, 4, 5);
  $x = array();
  var_dump(array_reduce($a, "sum")); // int(15)
  var_dump(array_reduce($a, "multiplication", 10));
// int(1200), because: 10*1*2*3*4*5
var_dump(array_reduce($x, "sum", "No data to reduce"));
// string(17) "No data to reduce"
?>
```

14.6 Searching for Values

The `in_array()` function searches an array for a specific value. If the third parameter **strict** is set to TRUE then the `in_array()` function will also check the types of the $value.

```
in_array( $value,  $array [,$strict ] );
```

//Example1

```
<?php
$os = array("windows", "redhat","Linux");
if(in_array("Linux", $os))
{
    echo "Got linux";
}
else
{
    echo "This os does not exists";
}
// with strict example
$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";
}
?>
```

A variation on `in_array()` is the `array_search()` function.

While `in_array()` returns `true` if the value is found, `array_search()` returns the key of the found element.

```
<?php
   $array = array(0 => 'vision',1 => 'publications',
       'bca'=>'sem1', 2 => 'sem2',3 => 'sem3');
     $key = array_search('sem1', $array);
//output => bca
   echo "output => $key\n";
   $key = array_search('vision', $array);    // output => 0
   echo "output => $key\n";
?>
```

The `array_search()` function also takes the optional third ***strict*** argument, which requires the types of the value being searched for and the value in the array to match.

15. Sorting

PHP comes with a number of built-in functions designed specifically for sorting array elements in different ways like alphabetically or numerically in ascending or descending order. Some of the most commonly used functions for sorting arrays are:

i. **sort() and rsort():** For sorting indexed arrays

ii. **asort() and arsort():** For sorting associative arrays by value

iii. **ksort() and krsort():** For sorting associative arrays by key

15.1 Sorting One Array at a Time

Sorting Indexed Arrays In Ascending Order

The sort() function is used for sorting the elements of the indexed array in ascending order (alphabetically for letters and numerically for numbers).

```
<?php
    $colors = array("potato", "tomato", "onion", "coriander");
    sort($colors);
    print_r($colors);
    $numbers = array(1, 2, 1.5, 4, 7, 10);
    sort($numbers);
```

```
    print_r($numbers);
?>
```

Output

Array ([0] => coriander [1] => onion [2] => potato [3] => tomato)
Array ([0] => 1 [1] => 1.5 [2] => 2 [3] => 4 [4] => 7 [5] => 10)

The rsort() function is used for sorting the elements of the indexed array in descending order (alphabetically for letters and numerically for numbers).

```
$colors = array("potato", "tomato", "onion", "coriander");
rsort($colors);
  //Array ([0] => tomato [1] => potato [2] => onion [3] => coriander)
$numbers = array(1, 2, 1.5, 4, 7, 10);
rsort($numbers);
//Array ([0] => 10 [1] => 7 [2] => 4 [3] => 2 [4] => 1.5 [5] => 1)
```

Sorting Associative Arrays in Ascending Order By Value

The asort() function sorts the elements of an associative array in ascending order according to the value. It works just like sort(), but it preserves the association between keys and its values while sorting.

```
<?php
$age = array("AAA"=>20, "BBB"=>14, "CCC"=>45, "DDD"=>35);
    asort($age);
  print_r($age);
?>
```

Output

Array([BBB] => 14 [AAA] => 20 [DDD] => 35 [CCC] => 45)

Sorting Associative Arrays in Descending Order By Value

The arsort() function sorts the elements of an associative array in descending order according to the value. It works just like rsort(), but it preserves the association between keys and its values while sorting.

```
$age = array("AAA"=>20, "BBB"=>14, "CCC"=>45, "DDD"=>35);
arsort($age);
Array ([CCC] => 45 [DDD] => 35 [AAA] => 20 [BBB] => 14 )
```

Sorting Associative Arrays in Ascending Order By Key

The ksort() function sorts the elements of an associative array in ascending order ascending to the key. It preserves the association between keys and its values while sorting, same as asort() function.

```
$age = array("AAA"=>20, "BBB"=>14, "CCC"=>45, "DDD"=>35);
 ksort($age);
Array ( [AAA] => 20 [BBB] => 14 [CCC] => 45 [DDD] => 35 )
```

Sorting Associative Arrays in Descending Order By Key

The krsort() function sorts the elements of an associative array in descending order ascending to the key.

It preserves the association between keys and its values while sorting, same as arsort() function.

```
$age = array("AAA"=>20, "BBB"=>14, "CCC"=>45, "DDD"=>35);
 krsort($age);
Array ( [DDD] => 35 [CCC] => 45 [BBB] => 14 [AAA] => 20 )
```

15.2 Natural-Order Sorting

PHP's built-in sort functions correctly sort strings and numbers, but they don't correctly sort strings that contain numbers. To correctly sort strings that contain numbers, use the `natsort()` and `natcasesort()` functions:

```
$output = natsort(input);
$output = natcasesort(input);
```

```
<?php
   $input1 = array('vision.txt', 'img1.txt', 'img0.txt', 'img2.txt',
  'img11.txt', 'IMG3.txt');
   $input2 = $input1;
   natsort($input2);
   echo " \n Natural order sorting \n";
   print_r($input2);
?>
```

Output
Natural order sorting

Array ([5] => IMG3.txt [2] => img0.txt [1] => img1.txt [3] => img2.txt [4] => img11.txt [0] => vision.txt)

natcasesort():Sorts an array using a case insensitive "natural order" algorithm.

```
<?php
   $array1 = array('ch0.txt', 'ch12.txt', 'ch10.txt', 'ch2.txt',
'ch1.txt', 'ch3.txt');
```

```
    $array2 = $array1;
    natcasesort($array2);
    echo " \n Natural order sorting (case-insensitive) \n";
    print_r($array2);
?>
```

Output

Natural order sorting (case-insensitive)
Array ([0] => ch0.txt [4] => ch1.txt [3] => ch2.txt [5] => ch3.txt [2] => ch10.txt [1] => ch12.txt)

15.3 Sorting Multiple Arrays at Once

array_multisort -- Sort multiple or multi-dimensional arrays

```
bool array_multisort(array ar1 [, mixed arg [, mixed ... [, array
...]]])
```

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

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.

The input arrays are treated as columns of a table to be sorted by rows - this resembles the functionality of SQL ORDER BY clause. The first array is the primary one to sort by. The rows (values) in that array that compare the same are sorted by the next input array, and so on.

The argument structure of this function is a bit unusual, but flexible. The first argument has to be an array. Subsequently, each argument can be either an array or a sorting flag from the following lists.

Sorting Order Flags

- `SORT_ASC:` Sort in ascending order
- `SORT_DESC:` Sort in descending order

Sorting type flags

- `SORT_REGULAR:` Compare items normally
- `SORT_NUMERIC:` Compare items numerically
- `SORT_STRING:` Compare items as strings

No two sorting flags of the same type can be specified after each array. The sorting flags specified after an array argument apply only to that array - they are reset to default ***SORT_ASC*** and ***SORT_REGULAR*** before each new array argument.

```
<?php
$a1=array("Dog","Cat");
$a2=array("Bruno","kitty");
array_multisort($a1,$a2);
print_r($a1);
print_r($a2);
```

```
//Output
Array ( [0] => Cat [1] => Dog ) Array ( [0] => kitty [1] => Bruno )
$a11=array("Dog","Dog","Cat");
$a22=array("Bruno","Duster","kitty");
array_multisort($a11,SORT_ASC,$a22,SORT_DESC);
print_r($a11);
print_r($a22);
//Output
Array ( [0] => Cat [1] => Dog [2] => Dog ) Array ( [0] => kitty [1] => Duster [2] =>
Bruno )
?>
```

15.4 Reversing Arrays

array_reverse – Returns an array with elements in reverse order.

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

array_reverse() takes input *array* and returns a new array with the order of the elements reversed, preserving the keys if *preserve_keys is TRUE.*

```
<?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 )
```

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

The `array_flip()` function returns an array that reverses the order of each original element's key-value pair:

```
$flipped = array_flip(array  trans);
```

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.

```
array_flip() returns FALSE if it fails.
<?php
   $a1=array("aa"=>"red","bb"=>"green","cc"=>"blue","dd"=>"yellow");
   $result=array_flip($a1);
   print_r($result);
?>
```

Output
Array ([red] => aa [green] => bb [blue] => cc [yellow] => dd)

15.5 Randomizing Order

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

```
shuffle( $array  );
```

This function assigns new keys for the elements in array. It removes any existing keys we may have assigned, rather than just reordering the keys.

It returns TRUE on success or FALSE on failure.

```
<?php
   $input = array("aaa"=>"lemon", "fff"=>"orange", "ccc"=>"banana" );
   shuffle($input);
   print_r($input);
?>
```

Output
Array([0] => orange [1] => lemon [2] => banana)

16. Using Arrays

In addition to arrays used for storing collections of values, arrays are also used to implement various abstract data types.

16.1 Sets

Arrays implement basic operations of set theory: union, intersection, and difference. Each set is represented by an array, and various PHP functions implement the set operations. The values in the set are the values in the array—the keys are not used, but they are generally preserved by the operations.

The `array_merge()` function to merge the elements or values of two or more arrays together into a single array. The merging is occurring in such a way that the values of one array are appended to the end of the previous array.

```
<?php
// Example 1 [Merging associative arrays. When two or more arrays have
//same key then the last array key value overrides the other one]

$array1 = array("a" => "JAVA", "b" => "ASP");
$array2 = array("c" => "C", "b" => "PHP");
print_r(array_merge($array1,$array2));

// Example 2 [merge arrays having integer keys and
//want to reset integer keys to start from 0 then use array_merge()
//function]

$array3 =array(5 => "CSS",6 => "CSS3");
$array4 =array(8 => "JAVASCRIPT",9 => "HTML");
print_r(array_merge($array3,$array4));

// Example 3 [ merge arrays having integer keys and
// want to retain integer keys as it is then use PLUS (+) operator to
//merge arrays]

$array5 =array(5 => "CSS",6 => "CSS3");
$array6 =array(8 => "JAVASCRIPT",9 => "HTML");
print_r($array5+$array6);
?>
```

Array_intersect(): The *intersection* of two sets is the set of elements they have in common. PHP's built-in `array_intersect()` function takes any number of arrays as arguments and returns an array of those values that exist in each. If multiple keys have the same value, the first key with that value is preserved.

```
array array_intersect(array $array1, array $array2[, array $array3...]);
```

The `array_diff()` function returns an array with values from the first array that are not present in the second.

```
array array_diff( array $array1, array $array2 [, array $array3 ...] );
```

It compares array1 against array2 and returns the difference.

```
<?php
   $f1 = array("aa" => "Banana", "Mango", "Chickoo");
   $f2 = array("bb" => "Banana", "Orange", "Mango");
   $result= array_intersect($f1, $f2);
   print_r($result);
   $d = array_diff($f1,$f2);
   print_r($d);
?>
Array ( [aa] => Banana [0] => Mango )
Array ( [1] => Chickoo )
```

16.2 Stacks

When working with arrays, we may often find ourself in a position where we need to add a few extra items to the array or remove some of the items we have already added. PHP has a few functions that are designed to help us with this process, allowing us to add new elements to the beginning or end of the array or to remove elements from the beginning or end of the array.

There are two simple ways to add an element to the end of an array in PHP. The first is to "push" the element onto the array.

array_push — Push one or more elements onto the end of array.

```
array_push(array,value1,value2...)
```

Example

```
<?php
   $a=array("a"=>"red","b"=>"green");
   array_push($a,"blue","yellow");
   print_r($a);
?>
// Array ( [a] => red [b] => green [0] => blue [1] => yellow )
```

The `array_pop()` function pops and returns the last value of the **array**, shortening the array by one element.

```
array_pop(array)
```

It returns the last value of the array, shortening the array by one element.

```
<?php
   $a=array("red","green","blue");
   array_pop($a);
```

```
  print_r($a);
?>
// Array ( [0] => red [1] => green )
```

There are also `array_shift()` and `array_unshift()` functions for treating an array like a queue.

Array_shift(): This function shifts the first value of the array off and returns it, shortening the array by one element and moving everything down.

```
array_shift ( $array );
```

It returns first element of the array and if array is empty (or is not an array), NULL will be returned.

```
<?php
   $input = array("orange", "banana");
   array_unshift($input, "apple", "pie");
   print_r($input);
    echo"remove one element";
   print_r("\n");
   print_r(array_shift($input));
   print_r("\n");
    echo"remove one element";
    print_r("\n");
   print_r(array_shift($input));
 print_r($input);
?>
Array ( [0] => apple [1] => pie [2] => orange [3] => banana )
remove one element apple
 remove one element pie
Array ([0] => orange [1] => banana)
```

Program

1. Write a menu driven program to perform the following stack related operations.

i. **Insert an element in stack.**

ii. **Delete an element from stack.[Hint: Use array_push(), array_pop()]**

HTML file

```
<html>
<body>
<form action="array.php" method="get">
<center>
```

```
   <h3><input type="text" name="str">
   <input type="radio" name="op" value="1">Insert element into
array</h3>
   <h3><input type="radio" name="op" value="2">Delete last element from
        array</h3>
   <h3><input type="radio" name="op" value="3">Display all elements
from        array</h3>
   <input type="submit" value="Submit">
</center>
</form>
</body>
</html>
```

Array.php file:

```
<?php
     $str = $_GET['str'];
     $op = $_GET['op'];
     $stack = array("mango", "banana", "apple", "strawberry");
     switch($op)
     {
       case 1 : array_push($stack,"$str");
          print_r($stack);
       break;
      case 2 : $last_elt = array_pop($stack);
            print_r($stack);
           print_r('deleted element is :'.$last_elt);
         break;
       case 3 : print_r($stack);
          break;
   }
?>
```

SUMMARY

- A function is a named block of code that performs a specific task, possibly acting upon a set of values given to it, or parameters, and possibly returning a single value.
- There are two basic types of functions. Built-in functions and user defined ones.
- Scope of variabel :Local variable,Global variables ,Static variables.
- Passing parameters to a function can be by value, and the other is by reference.
- PHP functions can return only a single value with the return keyword.
- Arrays are complex variables that allow us to store more than one value or a group of values under a single variable name.

- There are two kinds of arrays in PHP: indexed and associative.
- Associative arrays have strings as keys and behave more like two-column tables.
- The range() function creates an array containing a range of elements.
- The count() and sizeof() functions return the number of elements in the array.
- The array_pad() function inserts a specified number of elements, with a specified value, to an array.
- The list() function is used to assign values to a list of variables in one operation.
- The array_slice() function returns a new array consisting of a consecutive series of values from the original array.
- The array_chunk() function splits an array into chunks of new arrays.
- Array_keys() returns an array of all the keys in an array.
- array_values(): This function accepts a PHP array and returns a new array containing only its values (not its keys).
- The extract( ) function is used to import variables from an array into the current symbol table.
- The compact() function is used to create an array from variables and their values.
- The foreach loop statement allows us to iterate over elements of an array or public properties of an object.
- The iterator functions are: current(), reset(), next(), prev(), end(), each(), key().
- The array_walk() function runs each array element in a user-defined function.
- Some of the most commonly used functions for sorting arrays are :
- sort() and rsort(): sorting indexed arrays
- asort() and arsort(): sorting associative arrays by value
- ksort() and krsort(): sorting associative arrays by key
- To correctly sort strings that contain numbers, use the natsort() and natcasesort() functions.
- array_multisort() can be used to sort several arrays at once, or a multi-dimensional array by one or more dimensions.
- array_reverse: Returns an array with elements in reverse order.
- The array_flip() function returns an array that reverses the order of each original element's key-value pair.
- The array_merge() function merges the elements or values of two or more arrays together into a single array.

Exercise

1. What are different ways of passing parameters to function?
2. Explain different variable scope with reference to usage in function.
3. Explain variable function with example.
4. Explain anonymous function with suitable example.
5. What are missing parameters in php?
6. Differentiate between Indexed Vs Associative arrays.
7. How to declare an array in php?
8. How data is stored in arrays?
9. Explain the following functions: array_pad(), count(), sizeof(), list().
10. Explain multidimensional arrays with example.
11. Explain array_slice() with example.
12. Give different sorting functions used on array.
13. How can we reverse sort an array keeping the correlation between the index and value?
14. How to sort multidimensional array?
15. Which construct is used to traverse an array?
16. Explain array sorting functions.
17. Explain iterator functions in php.

Questions Asked in Previous Year Exams

1 Mark

A. Fill in the blanks

1. Subscript range of indexed array in PHP always starts with _____. [Apr. 2018]
   i. **0 (zero)** ii. 1
   iii. –1 iv. none
2. Which function sorts the array by keys in descending order? [Apr. 2018]
   i. **Ksort** ii. Krsort
   iii. Uksort iv. Uasort
3. What is the output of the following _____. [Apr. 2018]

```
<? php
$a = " 1E3 points of light" + 1,
echo "$a",
?>
```

   i. 1000 ii. 1001
   iii. 1E3 iv. **Error**

B. Answer the following

1. Which function identifies a subset of an array based on its values? [Apr. 2018]

3 Marks

1. Write a PHP program to find the elements from the array that matches the given value using appropriate search function. [Apr. 2018]
2. Write a menu driven program to perform the following operations on Associative arrays.
   i. Find the union of two arrays. ii. Find the intersection of two arrays.
   [Apr. 2018]

4 Marks

1. Define Anonymous function. Write anonymous function for addition of two integers. [Apr. 2018]
2. Write a PHP program to sort the array by values in ascending and descending order. Also sort the array by values without changing the keys. [Apr. 2018]
3. Describe Multidimensional array with example. [Apr. 2018]