Array

- An array is a single variable that can hold more than one value at once.
- You can think of an array as a list of values.
- Each value within an array is called an element, and each element is referenced by its own index, which is unique to that array.
- To access an elements value whether you are creating, reading, writing, or deleting the element you use that elements index.

Type of Array

 In PHP, there are three types of arrays:

1. Indexed arrays (Numeric arrays) - Arrays with a numeric index.
2. Associative arrays - Arrays with named keys.
3. Multidimensional arrays (Nested arrays) - Arrays containing one or more arrays.

1. Indexed arrays (Numeric arrays) :

- In numeric array each element having numeric key associated with it that is starting from 0.

Creating Arrays

- You can use array() function to create array.
- Syntax: $array_name=array (valuel, value2 ...valueN); There are two ways to create indexed arrays.
- The index can be assigned automatically (index always starts at 0), like this: $cars = array("Volvo", "BMW", "Toyota"); The index can be assigned manually:

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

Length of an Array

- The count() function is used to return the length (the number of elements) of an array.

```
<?php
    $cars = array("Volvo", "BMW", "Toyota"); echo
    count($cars);
?>
```

Accessing Array Elements:

- The print_r() function is used to print information about a variable.

```
<?php
    $myarray=array("A","B","C");
    print_r($myarray);
?> Output:
Array ( [0] => A [1] => B [2] => C )
```

- You can refer to individual element of an array in PHP script using its key value as shown below:

```
<?php
        Smyarray=array("A","B","C")
    ; echo $myarray[1]; ?>
```

- In Numeric Array you can use for, while or do while loop to iterate through each element in array because in numeric array key values are consecutive.

Changing Elements

- You can also change value of element using index.

```
<?php
        $myarray=array("Apache", "MySQL", "PHP");
        $myarray[1]="Oracle";           for($i=0;$i<3;$i++)

        {
                echo $myarray[$i]."<br>";
        }
?>
```

- Output:

```
Apache
Oracle
PHP
```

Add Elements to Array:

- The array_push() function inserts one or more elements to the end of an array.
- Syntax: array_push(array,value1,value2...)
    - **array:** Required. Specifies an array.
    - **Value1:** Required. Specifies the value to add. ▪ **value2:** Optional. Specifies the value to add.

```
<?php
        $myarray=array("Apache","MySQL", "PHP");
        print_r($myarray); echo
        "<br>";
        $myarray[]="Oracle";
        print_r($myarray);
        echo "<br>";
        array_push($myarray,"Java",".Net"); print_r($myarray);

?>
```

Output:

```
Array ([0] => Apache [1] => MySQL [2] => PHP)
Array ([0] => Apache [1] => MySQL [2] => PHP [3] => Oracle)
Array ([0] => Apache [1] => MySQL [2] => PHP [3] => Oracle [4] => Java [5] => .Net )
```

Removing Elements from Arrays:

- **Unset ()** is used to destroy a variable in PHP. It can be used to remove a single variable, multiple variables, or an element from an array.
- **Syntax:** unset (varl, var2....) var1, var2: The variable to be unset.
- You can remove the last element of an array using array_pop(). This will also return that element:

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

Output:

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

Searching Element:

- The array_search() function search an array for a value and returns the key.
- Syntax: array_search(value,array,strict) ○ value: Required. Specifies the value to search for. ○ array: Required. Specifies the array to search in.
  - ○ strict: Optional. If this parameter is set to TRUE, then this function will search for identical elements in the array.

```
<?php
        $a=array("A"=>"5","B"=>5);
        echo array_search(5,$a);
        echo array_search(5,$a,true);
        ?>
```

Output:

```
A B
```

- The **in_array()** function searches an array for a specific value.
- Syntax: in_array(search,array,type) ○ search: Required. Specifies the what to search for ○ array: Required. Specifies the array to search
  - ○ type: Optional. If this parameter is set to TRUE, the in_array() function searches for the specific type in the array.

```
<?php
        $people=array("Peter", "Joe", "Glenn", 23);  if
        (in_array("23",$people))
        {
```

```
		echo "Match found<br>";
	}
	else
	{
		echo "Match not found<br>";
	}
	if (in_array("23", $people, TRUE))
	{
		echo "Match found<br>";
	}
	else
	{
		echo "Match not found<br>";
	}
?>
```

Output:

```
	Match found
	Match not found
```

Sorting Array

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value krsort() - sort associative arrays in descending order, according to the key

```
<?php
	$srtArray=array(2,8,9,5,6,3);
	for ($i=0; $i<count($srtArray); $i++)
	{
		for ($j=0; $j<count($srtArray); $j++)
		{
			if ($srtArray[$j] > $srtArray[$i])
			{
				$tmp = $srtArray[$i];
				$srtArray[$i] = $srtArray[$j];
```

```
                                        $srtArray[$j] = $tmp;
                              }
                    }
          }
          foreach($srtArray as $item)
          {
                    echo Sitem."<br>\n";
          }
?>
Output:
2 3 5 6 8 9
```

Sort array using function. <?php

```
          $srtArray=array(2,8,9,5,6,3);
          foreach($srtArray as $item)
          {
                    echo $item."<br>\n";
          }
?>
Output:
2 3 5 6 8 9
```

2. Associative Array

- The associative part means that arrays store element values in association with key values rather than in a strict linear index order.
- If you store an element in an array, in association with a key, all you need to retrieve it later from that array is the key value.
- Key may be either numeric or string.
- You can use array() function to create associative array.
- Syntax: $array_name=array(key1=>value1, key1=>value1, keyN => valueN);
- There are two ways to create an associative array:
  $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
  OR
  $age["Peteru] = "35";
  $age["Ben"] = "37";
  $ager[“Joe”] = "43";
- You can refer to individual element of an array in PHP using its key value.

```
<?php
          $age = array("Peter"=>"35", "Ben"=>"37",
"Joe"=>"43");   echo "Ben:".$age["Ben"]; ?> Output:
Ben:37
```

- In associative array you cannot use for, while or do...while loop to iterate through each element in array because in Associative array key value are not consecutive.
- So you have to use foreach loop.

```
<?php
    $myarray=array("Name"=>"James", "Age"=>25,
    "Gender"=>"Male");    foreach($myarray as $item)  echo
    $item."<br>";
    }

?>
```

Output:

```
James
25
Male
```

3. Multidimensional Array

- Earlier, we have described arrays that are a single list of key/value pairs.
- However, sometimes you want to store values with more than one key.
- This can be stored in multidimensional arrays.
- A multidimensional array is an array containing one or more arrays.
- PHP understands multidimensional arrays that are two, three, four, five, or more levels deep.
- First, take a look at the following table:

Name	**Stock**	**Sold**
Volvo	22	18
BMW	15	13
Saab	5	2

```
<?php
    $cars = array (array("Volvo",22,18),
                    array("BMW",15,13), array("Saab",5,2));
```

Output:

```
    Volvo 22 18
    BMW 15 13
    Saab 5 2
```

User Defined Functions

- Besides the built-in PHP functions, we can create our own functions.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.

Create a User Defined Function

- A user defined function declaration starts with the word "function".

Syntax: function

```
function_name()
{
        Code to be executed;
}
```

- Example:

```
<?php  function writeMsg()

        {
                echo "Hello world!";
        }
        writeMsg();  // call the function
?>  Output:
Hello world!
```

Function arguments and returning values from function:

- Information can be passed to functions through arguments. An argument is just like a variable.
- Arguments are specified after the function name; inside the parentheses separate with comma.
- **Example:**

```
<?php function addFunction($num1,
    $num2)
    {
            $sum = $numl + $num2;
            return $sum;
    }
    $result=addFunction(10, 20);
    echo "Addition=".$result;
?>
Output:
Addition=30
```

Default values for arguments:

- You can specify default values for arguments.
- If the argument is omitted from the function call the default is used.
- The default value must be a constant expression such as a string, integer or NULL.
- You can have multiple arguments with default values. Default values assign from right to left.
- Example: <?php

```
    function addFunction($num1, $num2=5)
    {
            $sum = $numl + $num2;
```

```
                return $sum;
        }
        $result=addFunction(10,20);
        echo
        "Addition=".$result."<br>";
        $result=add Function(10); echo
        "Addition=".$result; ?> Output:
        Addition=30
        Addition=15
```

Call-by-Value (Pass-by-Value):

- The default behavior for user-defined functions in PHP is call-by-value.
- Call by value means passing the value directly to a function. The called function uses the value in a local variable; any changes to it do not affect the source variable.

```
<?php
function swap($numl, $num2)
{
        $temp = $num1;
        $numl=$num2;
        $num2=$temp;

}

        $numl=10; $num2=20;
        echo "Before Swap"."<br>";
        echo "numl=".$numl."<br>";
        echo
        "num2=".$num2."<br>";
        swap(Snum1,$num2); echo
        "After Swap"."<br>"; echo
        "numl=".$numl."<br>"; echo
        "num2=".$num2."<br>";
        ?>
```

Output:

```
        Before Swap numl=10
        num2=20 After
        Swap
        num1=10
        num2=20
```

Call-by-Reference (Pass-by-Reference):

- Call by reference means passing the address of a variable where the actual value is stored.

- The called function uses the value stored in the passed address; any changes to it do affect the source variable. <?php function swap(&$numl., &$num2)

```
                {
                $temp = Snuml; Snum1=Snum2;
                $num2=$temp;
        }
        $num1=10;
        $num2=20;
        echo "Before Swap"."<br>";
        echo "numl=".$numl."<br>";
        echo
        "num2=".$num2."<br>";
        swap($numl,$num2);  echo
        "After Swap"."<br>";
        echo "numl=".$numl."<br>"; echo "num2=".$num2."<br>";
        ?>
```

Output:

```
        Before Swap
        num1=10
        num2=20
        After Swap
        num1=20
        num2=10
```

Variables Scope:

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.
- PHP has three different variable scopes: ○ Local ○ Global ○ Static **Local:**

- A variable declared within a function has a local scope and can only be accessed within that function.

```
<?php function

myTest() {

  $x = 5; // local scope

  echo ''Variable x inside function is: $x";

}

myTest(); echo "Variable x outside function is: $x"; // using x outside the function will

generate an error ?>
```

Global:

- A variable declared outside a function has a global scope and can only be accessed outside a function:

```
<?php
    $x = 5; //global scope function
    myTest()
    {
      echo "Variable x inside function is: $x"; // using x inside this function will generate an error
    }
    myTest();
    echo "Variable x outside function is: $x";
?>
```

- The global keyword is used to access a global variable from within a function.
- To do this, use the global keyword before the variables (inside the function): **global $x; Static:**
- Normally, when a function is completed/executed, all of its variables are deleted.
- However, sometimes we want a local variable not to be deleted. We need it for a further job.
- To do this, use the static keyword when you first declare the variable:

```
<?php
function myTest()
{
      static $x = 0;    echo
      $x;
      $x++;
}
myTest(); echo
"<br>";
myTest();
echo "<br>";
myTest();
?>
```

```
Output:     0
            1
            2
```

 Then, each time the function is called, that variable will still have the information it contained from the last time the function called.

String Function:

Name	Description	Syntax:	Example	Output :
Chr	Returns a character from a specified ASCII value.	chr(ascii);	<?php echo chr(65); ?>	A

ord	Returns the ASCII value of the first character of a string.	ord(string);	<?php echo ord("A");
			echo ord("And");
			?>
				65
				65

Strtolower	Converts a string to lowercase letters.	strtolower(string)	<?php echo strtolower("HELLO"); ?>	hello
strtoupper	Converts a string to uppercase letters	strtoupper(string)	<?php echo strtoupper ("hello"); ?>	HELLO
strlen	Returns the length of a string.	strlen(string)	<?php echo strlen("hello hi"); ?>	8
ltrim	Remove whitespace / Characters from the left side of a string	ltrim(string,charlist)	<?php echo ltrim(“Hello”,"H"); echo ltrim(“ Hello”); ?>	ello Hello
	charlist: Optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed:"\O" — NULL, "\t" — tab, "\n" — new line, "\r" carriage return, " " — white space.			
rtrim	Remove whitespace / Characters from the right side of a string.	rtrim(string,charlist)	<?php echo rtrim(“Hello”,"o"); echo rtrim(“Hello ”); ?>	Hell Hello
	charlist:- Optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed:"\O" — NULL, "\t" — tab, "\n" — new line, "\r"—carriage return, " " — white space.			

trim	Remove whitespace / Characters from the both side of a string.	trim(string,charlist)	<?php
			echo trim("oHello","o");
			echo trim(“Hello”);
			?>
				Hell
				Hello

	charlist:- Optional. Specifies which characters to remove from the string If omitted, all of the following characters are removed:"\O" — NULL, "\t" — tab, "\n" — new line, "\r"— carriage return, " " — white space.			
substr	Returns a part of a string.)	echo substr("abcdef",-	substr(string,start,length<?php 2); ef echo substr("abcdef",-3,1); d substr("abcdef",0,-1); abcde echo substr("abcdef",2,-1); cde echo substr("abcdef",4,-4); ?>	echo
	start: Required. Specifies where to start in the string. **A positive number** - Start at a specified position in the string**. A negative number** - Start at a specified position from the end of the string. **0** - Start at the first character in string.			
	length: Optional. Specifies the length of the returned string. Default is to the end of the string. **A positive number** - The length to be returned from the start parameter**. Negative number** - The length to be returned from the end of the string.			

strcmp	Compares two strings (casesensitive).	strcmp(string1,string2)	<?php echo
			strcmp("hello","hello");
			echo strcmp("hello","hi");
			echo strcmp("hi","hello");
			?>
				0
				-1
				1

	strcmp function returns: 0 - if the two strings are equal, negative - if string1 is less than string2 and positive - if string1 is greater than string2.			
strcasecmp	Compares two strings (caseinsensitive)	strcasecmp(str1,str2)	<?php echo strcasecmp("hello","Hello"); ?>	0 0
	strcasecmp function returns: 0 - if the two strings are equal, negative - if string1 is less than string2 and positive - if string1 is greater than string2.			

strpos	Returns the position of the first occurrence of a string inside another string	strpos(string,find,start)	<?php echo strpos("hello","e"); echo strpos("hello","I"); ?>	1 2

	If the string is not found, this function returns FALSE. find:- Required. Specifies the string to find. start:- Optional. Specifies where to begin the search (case-sensitive).			
strrpos	Returns the position of the last occurrence of a string.	strrpos(string,find,start)	<?php echo strrpos("hello","l"); ?>	3
	If the string is not found, this function returns FALSE. find:- Required. Specifies the string to find. start:- Optional. Specifies where to begin the search. (case-sensitive).			

strstr	Finds the first occurrence of a string inside another string (case-sensitive).	strstr(string,search)
		echo strstr("Hello
		echo strstr("Hello
		?>
		Output:
			<?php
			world!","wor");
			world!","Wor");
			world!	

	This function returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found. search: Required. Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number.			
stristr	Finds the first occurrence of a string inside another string (caseinsensitive).	stristr(string,search)	<?php echo stristr("Hello world!","Wor"); ?> Output world!	
	This function returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found. x			
str_replace	Replaces some characters in a string (casesensitive).	str_replace(find,replace, string,count)	<?php echo str_replace(“world","Rahul","Hell o world!); ?> Output: Hello Rahul!	
	find: Required. Specifies the value to find replace: Required. Specifies the value to replace the value in find string: Required. Specifies the string to be searched count: Optional. A variable that counts the number of replacements			
strrev	Reverses a string.	strrev(string)	<?php echo strrev("hello"); ?>	olleh

Math Function:

Name	Description	Syntax	Example	Output
abs	Returns the absolute value of a number.	abs(x)	<?php echo abs(-6.7); echo abs(-3);	6.7 3

ceil	Returns the smallest integer value that is greater than or equal to a x.	ceil(x)	<?php
			echo ceil(0.60);
			echo ceil(5); echo ceil(-4.9);
			?>
				1
				5
				-4

floor	Returns the largest integer value that is less than or equal to a x.	floor(x)	<?php echo floor(0.60); echo floor(5);
			echo floor (4.9);
			?>
				0
				5
				-5

round	Rounds a number to the nearest integer.	round(x,precision)	<?php echo round(5.335,2); echo round(0.49); ?>	5.34 0
	precision: Optional. The number of digits after the decimal point			
fmod	Returns the remainder (modulo) of the division of the arguments.	fmod(x,y)	php echo fmod(5,2); ?>	1
min	Returns the number with the lowest value from specified numbers.	min(xl,x2,x3...)	<?php echo min(5,2,1,-4); ?>	-4
max	Returns the number with the highest value from specified numbers.	max(xl,x2,x3...)	<?php echo max(5,2,1,-4); ?>	5
pow	Returns the value of x to the power of y.	pow(x,y)	<?php echo pow(5,2); ?>	25
sqrt	Returns the square root of a number x.	sqrt(x)	<?php echo sqrt(25); ?>	5
rand	Returns a random integer	rand(min,max)	<?php echo rand(1,100); ?>	13

	min,max: Optional. Specifies the range the random number should lie within.
	If this function is called without parameters, it returns a random integer between 0 and RAND_MAX. (RAND_MAX depends on platform. For Windows value is 32768).

Date Functions:

Name	Syntax	Example	Output:
date	date(format,timestamp)	<?php echo date("d/m/y"); echo date("d M, Y"); ?>	11/08/15 11 Aug,
	Formats a local time/date. (**format:** Required, **timestamp:** optional) Specifie the format of date and time to be returned. d - The day of the month (from 01 to 31) m - A numeric representation of a month (from 01 to 12) M - A short textual representation of a month (three letters) Y - A four digit representation of a year y - A two digit representation of a year		
getdate	getdate(timestamp)	<?php print_r (getdate()); ?> Output: Array ([seconds] => 3 [minutes] => 49 [hours] => 17 [mday] => 11 [wday] => 2 [mon] => 8 [year] =>2015 [yday] => 222 [weekday] =>Tuesday [month] => August [0] => 1439308143)	
	Returns an array that contains date and time information for a UNIX timestamp. The returning array contains ten elements with relevant information needed when string: [seconds] — seconds, [minutes] — minutes, [hours] — hours, [mday] - day of the month, [wday]x formatting a date day of the week, [year] — year, [yday] - day of the year, [weekday] - name of the weekday, [month] - name of the month. **Timestamp:** Optional. Specifies the time in Unix time format		
checkdate	checkdate(month,day,year)	<?php var_dump(checkdate(2,29,2003)); var_dump(checkdate(2,29,2004)); ?>	boolean false boolean true
	The checkdate() function returns true if the specified date is valid, and false otherwise. month, day, year: Required. Specifies the month, day and year. A date is valid if: month is between 1. to 12 , day is within the allowed number of days for the particular month, year is between 1 to 32767.		

time	time(void)	<?php echo time(); ?>	143930992 2
	The time() function returns the current time as a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).		
mktime	mktime (hour,minute,second,month,day,ye ar,is_dst)	<?php echo (date("M-d Y",mktime(0,0,0,12,36,2001))); echo (date("M- d-Y",mktime(0,0,0,1,1,99))); ?> Output: Jan-05-2002 Jan-01-1999	
	The mktime() function returns the Unix timestamp for a date. This timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified. **hour, minute, second, month, day, year:** Optional. Specifies the hour, minute, second, month, day, year. **is _dst**: Optional. Set this parameter to 1 if the time is during daylight savings time (DST), 0 if it is not, or -1 (the default) if it is unknown.		

File Function:

Name	Syntax	Example	O/P
fwrite0	fwrite(file,string,length)	<?php $file = fopen("test.txt","w"); echo fwrite($file,"Hello World Testing!"); fclose($file); ?>	21
	The fwrite() writes to an open file. The function will stop at the end of the file or when it reaches the specified length, whichever comes first. This function returns the number of bytes written, or FALSE on failure. This function is binary-safe.		
	file: Required. Specifies the open file to write to string: Required. Specifies the string to write to the open file length: Optional. Specifies the maximum number of bytes to write		
fopen()	fopen(filename,mode,include_path,context)	<?php $file = fopen("test.txt","r"); $file = fopen("/home/test/test.txt","r"); $file = fopen("http://www.example.com/","r"); ?>	
	The fopen() function opens a file or URL. If fopen() fails, it returns FALSE and an error on failure. The file may be opened in one of the following modes: r,w,a,x,r+,w+,a+,x+		

fclose()	fclose(file)	<?php
		$myfile = fopen("webdictionary.txt", "r"); // some code to be executed....
		fclose($myfile);
		?>

	The fclose() function is used to close an open file. The fclose() requires the name of the file (or a variable that holds the filename).	