Parul® University

NAAC A++ ACCREDITED UNIVERSITY

Presentation

UNIT -2 : OVERVIEW OF PHP STRUCTURE AND SYNTAX

Prepared By Prof. Aneri Patel

BACKGROUND INFORMATION OF PHP

introduction

Structure

Features

INTRODUCTION

- PHP was created by **Rasmus Lerdorf in 1994** but appeared in the market in 1995.
- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use
- PHP creates dynamic websites.
- PHP can be embedded directly into the HTML code.
- PHP supports many databases like MySQL, Oracle, etc.
- PHP is an interpreted language, i.e. there is no need for compilation.
- PHP is an object oriented language.

INTRODUCTION

What is PHP file?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

INTRODUCTION

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

INTRODUCTION

Why use php?

- It runs on different platforms such as Windows, Linux, Unix, etc.
- This language is very simple to learn and runs efficiently on the server side.
- It is compatible with almost all servers used today, such as Apache, IIS, etc.
- It supports many databases such as MySQL, Oracle, PostgreSQL etc.
- It is embedded directly into the HTML code.
- PHP can also be used to create dynamic web pages.
- It is often used together with Apache (web server) on various operating systems. It can be also used with Microsoft's IIS on Windows.
- It is open source and it is free downloadable

STRUCTURE

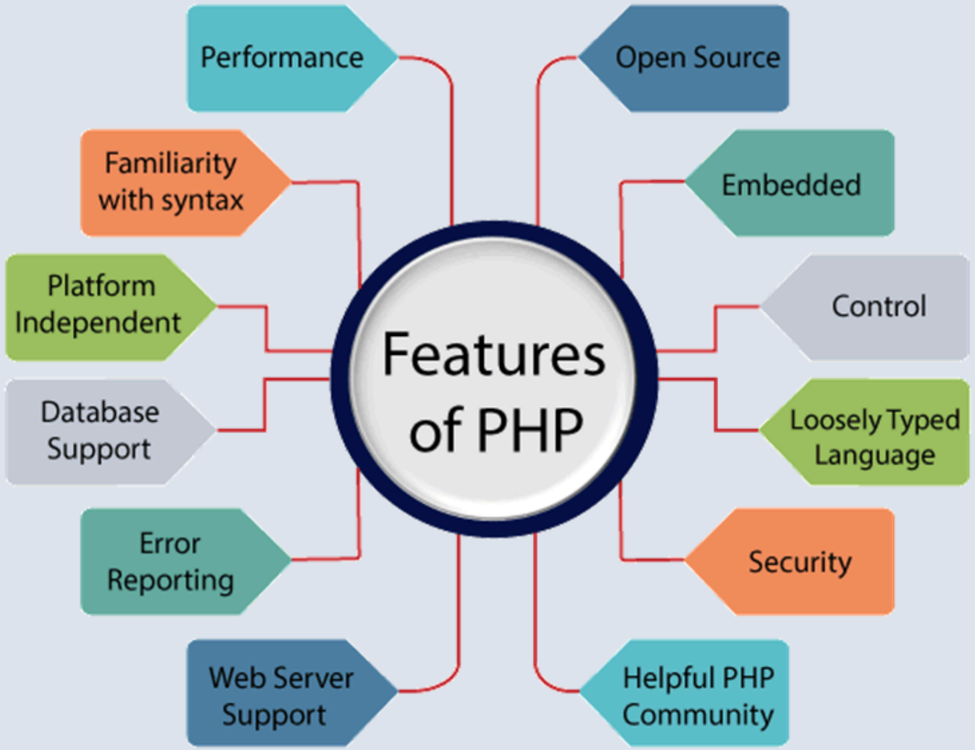
Basic Syntax of PHP

- PHP code is start with <?php and ends with ?>
- Every PHP statements end with a semicolon (;).
- PHP code save with .php extension.
- PHP contain some HTML tag and PHP code.
- You can place PHP code any where in your document.

STRUCTURE

```
<html>
    <head>
        <title>PHP Structure</title>
    </head>
    <body>
        <?php
            //Php Code
        ?>
    </body>
</html>
```

FEATURES OF PHP



FEATURES OF PHP

- **Simple:** It is very simple and easy to use, compare to other scripting language
- **Interpreted :** It is an interpreted language, i e there is no need for compilation
- **Faster:** It is faster than other scripting language eg. asp
- **Open Source:** Open source means you no need to pay for use php, you can free download and use
- **Platform Independent:** PHP code will be run on every platform, Linux, Unix, Mac OS, Windows

FEATURES OF PHP

- **Case Sensitive:** PHP is case sensitive scripting language at time of variable declaration In PHP, all keywords (e g if, else, while, echo, etc Classes, functions, and user defined functions are NOT case sensitive
- **Performance:** faster than those written in other scripting languages
- **Portability:** PHP runs on various platforms (Linux, Unix, Mac OS X, etc
- **Ease of Use:** 5000 functions included with the core distribution

FEATURES OF PHP

- **Loosely Typed Language:** PHP supports variable usage without declaring its data type It will be taken at the time of the execution based on the type of data it has on its value
- **Embedded:** PHP code can be easily embedded within HTML tags and script
- **Compatibility:** PHP is compatible with almost all local servers used today like Apache,IIS etc
- **Error Reporting:** PHP have some predefined error reporting constants to generate a warning or error notice

COMMENTS ON PHP

```
// This is a single-line comment

# This is also a single-line comment

/* This is a
multi-line comment */
```

ECHO AND PRINT STATMENT

- In PHP there are two basic ways to get output: echo and print.
- echo and print are more or less the same. They are both used to output data to the screen.
- The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

CONSTANT

introduction

CONSTANT INTRODUCTION

- Constants are like variables except that once they are defined they cannot be changed or undefined.
- A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no $ sign before the constant name).
- Note: Unlike variables, constants are automatically global across the entire script.

SYNTAX OF CONSTANT USING DEFINE FUNCTION

To create a constant, use the define() function :

Syntax: define(name, value);

Example:

```
echo "constant using define : \t\t";
define("GREETING", "Welcome to PHP");
echo GREETING;
```

SYNTAX OF CONSTANT USING DEFINE FUNCTION

constant array

```
echo "constant php array : \t\t";
define("course", [
  "BCA",
  "MCA",
  "B.sc(IT)",
  "M.sc(IT)"
]);
echo course[1];
```

SYNTAX OF CONSTANT USING DEFINE FUNCTION

constant used in function

```
echo "constant are global : \t\t";
function con1() {
  echo GREETING;
}
con1();
```

SYNTAX OF CONSTANT USING CONST KEYWORD

- You can also create a constant by using the const keyword.
- const cannot be created inside another block scope, like inside a function or inside an if statement.
- **Syntax:** const name = value;
- **Example:**

```
echo "constant using const : \t\t";
const PI = 3.14;
echo PI;
```

USING VARIABLES

Types

Rules

Example

DATA TYPE

A Data type is the classification of data into a category according to its attributes;

- Alphanumeric characters are classified as strings
- Whole numbers are classified integers
- Numbers with decimal points are classified as floating points.
- True or false values are classified as Boolean.

DATATYPE

PHP is a loosely typed language; it does not have explicit defined data types. PHP determines the data types by analyzing the attributes of data supplied. PHP implicitly supports the following data types:

- Scalar types : boolean, integer, float, string
- Compound types: array, object
- Special types: resources, NULL

VARIABLES

- **Integers** – are whole numbers, without a decimal point, like
- **Doubles** – are floating point numbers, like 3.14159 or 49.1.
- **Booleans** – have only two possible values either true or false
- **NULL** – is a special type that only has one value: NULL
- **Strings** – are sequences of characters, like 'PHP supports string operations'.
- **Arrays** – are named and indexed collections of other values.
- **Objects** – are instances of programmer defined classes, which can package up both otherkinds of values and functions that are specific to the class.
- **Resources** – are special variables that hold references to resources external to PHP (such asdatabase connections).

RULES FOR VARIABLE DECLARATION

- Must start with letter or underscore.
- Never use numeric value as an initial character of variable.
- No need to defined datatype while declaring variable.
- Use $ sign for declaration.
- Variable names are case sensitive.
- A variable name can only contain alpha numeric characters and underscores (A z, 0 9, and _)

  $a $_1345 $ABC
  $NFC $_ab
  $abc
  $1234 $_123&*() $$.

EXAMPLE

```
<?php
$str="Hello world!";
$a=5;
$b=10.5;
echo "String is: $str <br />";
echo "Integer is: $x <br />";
echo "Float is: $y <br />";
?>
```

EXAMPLE

```
# variable declaration
$name = "Aneri";
$year = 2024;
echo "<h2> PHP Variable Demo </h2>";
echo $name , $year;
echo "<br>";
print $year;

#using single quotes
echo '<h3> My Name is '.$name.'</h3>';
```

VARIABLE TYPE CASTING

Introduction

VARIABLE TYPE CASTING

- Type casting is converting a variable or value into a desired data type.
- This is very useful when performing arithmetic computations that require variables to be of the same data type.
- Type casting in PHP is done by the interpreter.

VARIABLE TYPE CASTING

- PHP also allows you to cast the data type. This is known as explicit casting. The code below demonstrates explicit type casting.

```
<?php
    $a = 1;
    $b = 1.5;
    $c = $a + $b
    $c = $a + (int) $b
    echo $c;
?>
```

OPERATORS

Introduction

Types

Example

INTRODUCTION

- PHP Operator is a symbol i.e used to perform operations on operands. In simple words, operators are used to perform operations on variables or values. For example:
- $num=10+20;//+ is the operator and 10,20 are operands
- In the above example, + is the binary + operator, 10 and 20 are operands and $num is variable.
- PHP Operators can be categorized in following forms:

LIST OF OPERATORS IN PHP

- Arithmetic operators
- Assignment operators
- Bitwise operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional assignment operators

ARITHMETIC OPERATORS

Operator	Name	Example	Explanation
+	Addition	$a + $b	Sum of operands
-	Subtraction	$a - $b	Difference of operands
*	Multiplication	$a * $b	Product of operands
/	Division	$a / $b	Quotient of operands
%	Modulus	$a % $b	Remainder of operands
**	Exponentiation	$a ** $b	$a raised to the power $b

ARITHMETIC OPERATORS

Example:

```
<?php
$x = 15;
 $y = 10;

echo $x + $y;
?>
```

ASSIGNMENT OPERATORS

Operator	Name	Example	Explanation
=	Assign	$a = $b	The value of right operand is assigned to the left operand.
+=	Add then Assign	$a += $b	Addition same as $a = $a + $b
-=	Subtract then Assign	$a -= $b	Subtraction same as $a = $a - $b
*=	Multiply then Assign	$a *= $b	Multiplication same as $a = $a * $b
/=	Divide then Assign (quotient)	$a /= $b	Find quotient same as $a = $a / $b
%=	Divide then Assign (remainder)	$a %= $b	Find remainder same as $a = $a % $b

ASSIGNMENT OPERATORS

Example 1:

```
<?php
$x = 10;
echo $x;
?>
```

Example 2:

```
<?php
$x = 20;
$x += 100;

echo $x;
?>
```

BITWISE OPERATORS

Operator	Name	Example	Explanation
&	And	$a & $b	Bits that are 1 in both $a and $b are set to 1, otherwise 0.
|	Or (Inclusive or)	$a | $b	Bits that are 1 in either $a or $b are set to 1
^	Xor (Exclusive or)	$a ^ $b	Bits that are 1 in either $a or $b are set to 0.
~	Not	~$a	Bits that are 1 set to 0 and bits that are 0 are set to 1
<<	Shift left	$a << $b	Left shift the bits of operand $a $b steps
>>	Shift right	$a >> $b	Right shift the bits of $a operand by $b number of places

COMPARISON OPERATORS

Operator	Name	Example	Explanation
==	Equal	$a == $b	Return TRUE if $a is equal to $b
===	Identical	$a === $b	Return TRUE if $a is equal to $b, and they are of same data type
!==	Not identical	$a !== $b	Return TRUE if $a is not equal to $b, and they are not of same data type
!=	Not equal	$a != $b	Return TRUE if $a is not equal to $b
<>	Not equal	$a <> $b	Return TRUE if $a is not equal to $b
<	Less than	$a < $b	Return TRUE if $a is less than $b
>	Greater than	$a > $b	Return TRUE if $a is greater than $b
<=	Less than or equal to	$a <= $b	Return TRUE if $a is less than or equal $b
>=	Greater than or equal to	$a >= $b	Return TRUE if $a is greater than or equal $b
<=>	Spaceship	$a <=>$b	Return -1 if $a is less than $b Return 0 if $a is equal $b Return 1 if $a is greater than $b

INCREMENTING/DECREMENTING OPERATORS

Operator	Name	Example	Explanation
++	Increment	++$a	Increment the value of $a by one, then return $a
		$a++	Return $a, then increment the value of $a by one
--	decrement	--$a	Decrement the value of $a by one, then return $a
		$a--	Return $a, then decrement the value of $a by one

LOGICAL OPERATORS

Operator	Name	Example	Explanation
and	And	$a and $b	Return TRUE if both $a and $b are true
Or	Or	$a or $b	Return TRUE if either $a or $b is true
xor	Xor	$a xor $b	Return TRUE if either $ or $b is true but not both
!	Not	! $a	Return TRUE if $a is not true
&&	And	$a && $b	Return TRUE if either $a and $b are true
||	Or	$a || $b	Return TRUE if either $a or $b is true

LOGICAL OPERATORS

Example :

```
<?php
$x = 100;
 $y = 60;

if ($x == 100 && $y == 50) {
 echo "Hello world!";
 }
 else
 {
echo "not Matched";
 }
?>
```

STRING OPERATORS

Operator	Name	Example	Explanation
.	Concatenation	$a . $b	Concatenate both $a and $b
.=	Concatenation and Assignment	$a .= $b	First concatenate $a and $b, then assign the concatenated string to $a, e.g. $a = $a . $b

STRING OPERATORS

Example:

```
<?php
$txt1 = "Hello";
$txt2 = " world!";
echo $txt1 . $txt2;
?>
```

ARRAY OPERATORS

Operator	Name	Example	Explanation
+	Union	$a + $y	Union of $a and $b
==	Equality	$a == $b	Return TRUE if $a and $b have same key/value pair
!=	Inequality	$a != $b	Return TRUE if $a is not equal to $b
===	Identity	$a === $b	Return TRUE if $a and $b have same key/value pair of same type in same order
!==	Non-Identity	$a !== $b	Return TRUE if $a is not identical to $b
<>	Inequality	$a <> $b	Return TRUE if $a is not equal to $b

ARRAY OPERATORS

Example:

```
<?php
$x = array("a" => "red", "b" => "green");
$y = array("c" => "blue", "d" => "yellow");

print_r($x + $y); // union of $x and $y
?>
```

CONDITIONAL ASSIGNMENT OPERATORS

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

CONDITIONAL STATEMENTS

Introduction

Syntax

Example

CONDITIONAL STATEMENT

- Conditional statements are used to perform different actions based on different conditions.
- In PHP we have the following conditional statements:
  - if statement - executes some code if one condition is true
  - if...else statement - executes some code if a condition is true and another code if that condition is false
  - if...elseif...else statement - executes different codes for more than two conditions
  - switch statement - selects one of many blocks of code to be executed

IF STATEMENT

The if statement executes some code if one condition is true.

Syntax:

```
if (condition) {
// code to be executed if condition is true;
}
```

IF STATEMENT

Example:

```
<?php
$x = 12;

if ($x > 0)
{
echo "The number is positive";
}
?>
```

IF...ELSE STATEMENTS

The if...else statement executes some code if a condition is true and another code if that condition is false.

Syntax:

```
if (condition)
{
// code to be executed if condition is true;
}
else
{
// code to be executed if condition is false;
}
```

IF...ELSE STATEMENTS

Example:

```
<?php
$a=10;
$b=20;
if ($a > $b)
echo "a is bigger than b";
else
echo "a is not bigger than b";
?>
```

IF...ELSEIF...ELSE STATEMENT

The if...elseif...else statement executes different codes for more than two conditions.

Syntax:

```
if (condition)
{
code to be executed if this condition is true;
}
elseif (condition)
{
// code to be executed if first condition is false and this condition is true;
}
else
{
// code to be executed if all conditions are false;
}
```

IF...ELSEIF...ELSE STATEMENT

Example:

```
<?php
$d = date("D");
if ($d == "Fri")
echo "<h3>Have a nice weekend!</h3>";
elseif ($d == "Sun")
echo "<h3>Have a nice Sunday!</h3>";
else
echo "<h3>Have a nice day!</h3>";
?>
```

SWITCH STATEMENT

Use the switch statement to select one of many blocks of code to be executed.

Syntax:

```
switch (expression)
{
case label1:
//code block
break;
case label2:
//code block;
break;
case label3:
//code block
break;
default:
//code block
}
```

SWITCH STATEMENT

```
<?
php
switch($category) {
case "news":
echo "<p>What's happening around the world</p>";
break;
case "weather":
echo "<p>Your weekly forecast</p>";
break;
case "sports":
echo "<p>Latest sports highlights</p>";
break;
default:
echo "<p>Welcome to my web site</p>";
}
```

LOOPING

Introduction

Syntax

Example

LOOP

- Loops are used to repeatedly execute the same block of code as long as a condition is met. The basic concept behind a loop is to save time and effort by automating repetitive activities within a program. There are four different types of loops supported by PHP.

TYPES OF LOOPS

- while - loops through a block of code as long as the specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array

WHILE LOOP

- The while loop - Loops through a block of code as long as the specified condition is true.

- Syntax:

```
while (if the condition is true)
{
// code is executed
}
```

WHILE LOOP

```php
<?php
$i = 1;

while ($i < 6) {
  echo $i;
  $i++;
}

?>
```

DO....WHILE LOOP

- The do...while loop will always execute the block of code at least once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax :

```
do
{
//code is executed
}
while (if condition is true);
```

DO....WHILE LOOP

```
<?php

// PHP code to illustrate do...while loops
$num = 2;
do {
$num += 2;
echo $num, "\n";
} while ($num < 12);

?>
```

Output: 4 6 8 10 12

FOR LOOP

- The for loop - Loops through a block of code a specified number of times.
- The for loop is used when you know how many times the script should run.
-
- Syntax:

```
for (expression1, expression2, expression3)
 {
// code block
}
```

FOR LOOP

```php
<?php
for ($x = 0; $x <= 10; $x++) {
  echo "The number is: $x <br>";
}
?>
```

FOR EACH STATEMENT

- The foreach loop - Loops through a block of code for each element in an array or each property in an object.
- The most common use of the foreach loop, is to loop through the items of an array.
- Syntax:

```
foreach (array_element as value)
 {
  //code to be executed
}
```

FOR EACH STATEMENT

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {
  echo "$x <br>";
}
?>
```

ARRAY

Introduction

Types

Function

ARRAY

- An array is a special variable that we use to store or hold more than one value in a single variable.
- We can store number, string and object in the PHP array.
- All PHP array elements are assigned to an index number by default.
- There are different types of arrays in PHP. They are:
  - Numeric array – PHP index is represented by number which starts from 0.
  - Associative array – An array with strings as index. This stores element values in association with key
  - Multidimensional array – An array containing one or more arrays and values.

NUMERIC ARRAY

- A numeric array stores each element with a numeric ID key.
- 2 ways to write a numeric array.
  - Automatically
  - Manually

1. Automatically

   Example: $numbers = array(1, 2, 3, 4, 5);

2. Manually
   - Example:

```
$numbers[0] = 1;
$numbers[1] =2;
$numbers[2] = 3;
$numbers[3] = 4;
$numbers[] = 5; // to append value in an array.
```

NUMERIC ARRAY

Access The values

- To access specific value

```
echo $
numbers[0 ] . " and " . numbers[1 ]
```

- To access all values:

```
foreach
( $numbers as $value )
echo "Value is $value <
br />";
}
```

ASSOCIATIVE ARRAYS

- Associative array will have their index as string
- An associative array, each ID key is associated with a value
- When storing data about specific named values, a numerical array is not always the best way to do it
- With associative arrays we can use the values as keys and assign values to them

ASSOCIATIVE ARRAYS

- First Way

$variable= array(“Key1" => value1, " Key2" => value2, “Key3" => value3);

Example

$salaries = array(“ONDC" => 2000, “Swiggy " => 1000, “Zomato" => 500);

- Second way :

$salaries[‘ONDC '] = "high";

$salaries[‘Swiggy '] = "medium”;

$salaries[‘Zomato '] = "low”;

MULTIDIMENSIONAL ARRAY

- In a multidimensional array, each element in the main array can also be an array
- Each element in the sub array can be an array, and so on
- Example:
- $AmazonProducts =

```
array(“BOOK", "Books", 50),
array("DVDs", “Movies", 15),
array(“CDs", “Music", 20)
);
```

MULTIDIMENSIONAL ARRAY

Accessing multidimensional array

Example:

```
for ($row = 0; $row < 3; $row++) {
for ($column = 0; $column < 3; $column++)
{
    echo $AmazonProducts [$row][$column] . “ <BR>”;
}
}
```

MULTIDIMENSIONAL ARRAY

```
$AmazonProducts = array(
array(“Code” =>“BOOK", “Description” => "Books", “Price” => 50),
array(“Code” => "DVDs",“Description ” => Movies",“Price ” =>
array(“Code” => “CDs", “Description” => “Music", “Price” => 20)
);
for ($row = 0; $row < 3; $row++)
{
echo $AmazonProducts [$row][“Code”]. “ ” .
$AmazonProducts [$row][“Description”]. “ ” .
$AmazonProducts [$row][“Price”];
}
```

ARRAY FUNCTION

array_keys() : Returns all the keys of an array

Example:

```
<?php
$a=array("Volvo"=>"XC90","BMW"=>"X5","Toyota"=>"Highlander");
print_r(array_keys($a));
?>
```

array_merge() : Merges one or more arrays into one array

```
<?php
$a1=array("red","green");
$a2=array("blue","yellow");
print_r(array_merge($a1,$a2));
?>
```

ARRAY FUNCTION

array_pop(): Deletes the last element of an array

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

array_push(): Inserts one or more elements to the end of an array

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

ARRAY FUNCTION

array_reverse() : Returns an array in the reverse order

```
<?php
$a=array("a"=>"Volvo","b"=>"BMW","c"=>"Toyota");
print_r(array_reverse($a));
?>
```

array_search() : Searches an array for a given value and returns the key

```
<?php
$a=array("a"=>"red","b"=>"green","c"=>"blue");
echo array_search("red",$a);
?>
```

ARRAY FUNCTION

array_slice(): Returns selected parts of an array

```
<?php
$a=array("red","green","blue","yellow","brown");
print_r(array_slice($a,2));
?>
```

array_sum(): Returns the sum of the values in an array

```
<?php
$a=array(5,15,25);
echo array_sum($a);
?>
```

ARRAY FUNCTION

current(): Returns the current element in an array.

```
$people =array("Peter", Peter"," Joe"," Glenn","
echo current($people)
```

end(): Sets the internal pointer of an array to its last element.

```
$people =array("Peter", Peter"," Joe"," Glenn", “Cleveland");
echo current($people) .”<br>”;
echo end($people)
```

next(): Advance the internal array pointer of an array.

```
$people = array("Peter", "Joe", "Glenn", "Cleveland");
echo current($people) . "<br>”;
echo next($people);
```

ARRAY FUNCTION

array_values(): Returns all the values of an array

```
<?php
$a=array("Name"=>"Peter","Age"=>"41","Country"=>"USA");
print_r(array_values($a));
?>
```

count(): Returns the number of elements in an array

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

ARRAY FUNCTION

sort($array): Sorts an indexed array in ascending order.

```
$cars array("Volvo", Volvo," BMW","
sort($cars);
```

asort ($array) : Sorts an associative array in ascending order, according to the value.

```
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
asort ($age);
```

ksort ($array): Sorts an associative array in ascending order, according to the key.

```
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
ksort ($age);
```

ARRAY FUNCTION

rsort ($array) Sorts an indexed array in descending order

```
$cars=array(" Volvo","BMW","Toyota
rsort ($car);
```

arsort ($array) : Sorts an associative array in descending order, according to the value

```
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
arsort ($age);
```

krsort ($array) : Sorts an associative array in descending order, according to the key

```
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
krsort ($age);
```