**PHP**

**NOTES**

**INDEX**

* Introduction
* Features
* Installation
* First Program
* Variables
* Operators
* Decision Making/Conditional Statement
* While Loop
* Do-While Loop
* For Loop, For each
* Nested For Loop
* Break statement
* Continue statement
* Functions
* Array
* PHP Functions For Sorting Arrays
* String
* String functions
* Form Handling
* Php Include and require
* Php Date and Time Functions
* Cookies
* Sessions
* Crud Operation

PHP

**Introduction**

* PHP (Hypertext Preprocessor) is an open-source HTML-embedded server-side scripting language, which is used to develop dynamic and interactive web applications.
* PHP is developed by the Rasmus Lerdorf in 1995
* PHP is a server-side scripting language. Server-side scripting means that the PHP code is processed on the web server rather than the client machine.
* PHP page is a file with .php extension that contains could be combination of HTML Tags and PHP scripts.
* PHP is an interpreted language, i.e. there is no need for compilation.
* PHP is an object-oriented language.
* PHP is an open-source scripting language.
* PHP is simple and easy to learn language.
* PHP is free to download and use.
* PHP supports a wide range of databases.
* PHP recursive acronym for PHP(Hypertext Preprocessor): HyperText means, text containing all sorts of web markups, Pre Processor means all of the HyperText is processed first and then the result is send as pure HTML to the web browser. A client cannot see the PHP source code because it is pre processed and interpreted.

**Features:**

* **Performance: Its performance is faster than other scripting language.**
* **Compatibility: PHP is compatible with almost all servers used today (Apache, IIS, etc.)**
* **Platform independent: PHP are available for WINDOWS, MAC, and LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.**
* **Simplicity**
* **Flexibility**

**Common uses of PHP**

* 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 encrypt data.

**Installation:**

You can separately install Web Server, PHP Interpreter and MySQL databases, but to make work easier, developers has made all in one setup package called WAMP, LAMP, MAMP and XAMPP which will automatically install and setup PHP environment on your Windows, Linux or MAC machines.

**WAMP (Windows, Apache , MySQL, PHP)**

**LAMP (Linux, Apache , MySQL, PHP)**

**MAMP (MAC , Apache , MySQL, PHP)**

**XAMPP (Windows/Linux/MAC, Apache , MySQL, PHP)**

**First program:**

Print Hello World in php??

<!DOCTYPE>

<html>

<body>

<? php

echo "<h2>Hello First PHP</h2>";

print "<h2>Hello First PHP</h2>";

?>

</body>

</html>

**Echo:**

* echo is a statement i.e used to display the output. it can be used with parentheses echo or without parentheses echo.
* echo can pass multiple string separated as ( , )
* echo doesn’t return any value
* echo is faster then print

**Print:**

* Print is also a statement i.e used to display the output. it can be used with parentheses print( ) or without parentheses print.
* using print can doesn’t pass multiple argument
* print always return 1
* it is slower than echo.

**Variables**

* The main purpose of variables is to store data in memory for later use.
* variables value may change during execution.
* We can make any type of variable, like numbers, text strings and arrays.
* All variables in PHP starts with a $ dollar sign and the value can be assigned by using the = assignment operator.
* PHP Variables are case sensitive, variables always defined with a $, and they must start with an underscore or a letter (no number)
* In PHP unlike other languages, variables do not have to be declared before assign a value.
* You also don’t require to declare data types, because PHP automatically converts variable to data types depends upon its value.

**Variable name can not contents space.**

<?php

$me = "I am David"; // String type data

echo $me;

$num = 24562; // Number (integer)

echo $num;

$a = 2.5; // Float

$name = "David"; //Valid variable name

$\_name = "Alex"; //Valid variable name

$1name = "Jhon"; //Invalid variable name, starts with a number

?>

<?php

$str="hello string";

$x=200;

$y=44.6;

$4c="hello"; //number (invalid)

$\*d="hello"; //special symbol (invalid)

echo "string is: $str <br/>";

echo "integer is: $x <br/>";

echo "float is: $y <br/>";

?>

PHP is a loosely typed language, it means PHP automatically converts the variable to its correct data type.

**PHP Variable: Sum of two variables**

<?php

$a=5;

$b=6;

$c=$a+$b;

echo $c;

?>

**PHP Variable: case sensitive**

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COLor etc.

<?php

$color="red";

echo "My car is " . $color . "<br>";

echo "My house is " . $COLOR . "<br>";

echo "My boat is " . $coLOR . "<br>";

?>

**PHP Operators**

PHP Operator is a symbol i.e used to perform operations on operands. For example:

$num=10+20;//+ is the operator and 10,20 are operands

PHP Operators can be categorized in following forms:

* Arithmetic operators ( +,-,/,\*,%)
* Assignment operators ( =, +=)
* Comparison operators ( ==, >=, <= , != , >, < )
* Increment/Decrement operators ( ++, --)
* Logical operators ( &&, ||, !)
* String operators

**PHP Arithmetic Operators**

There are five basic arithmetic operators.

* + (addition)
* -(Subtration)
* (multiplication)
* / (division)
* % (modulus)

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Example | Result |
| + | Addition | $a+$b | Sum of $a and $b |
| - | Subtration | $a-$b | Difference of $a and $b |
| \* | Multiplication | $a\*$b | Product of $a and $b |
| / | Division | $a/$b | Quotient of $a and $b |
| % | Modulus | $a%$b | Reminder of $a and $b |

**Example:**

<?php

$x=100;

$y=60;

echo "The sum of x and y is : ". ($x+$y) ."<br />";

echo "The difference between x and y is : ". ($x-$y) ."<br />";

echo "Multiplication of x and y : ". ($x\*$y) ."<br />";

echo "Division of x and y : ". ($x/$y) ."<br />";

echo "Modulus of x and y : " . ($x%$y) ."<br />";

?>

**Output:**

The sum of x and y is : 160

The difference between x and y is : 40

Multiplication of x and y : 6000

Division of x and y : 1.6666666666667

Modulus of x and y : 40

PHP Assignment Operators

Assignment operators allow writing a value to a variable. The first operand must be a variable and the basic assignment operator is "=".

All five arithmetic operators have corresponding assignment operators, Here is the list.

|  |  |  |
| --- | --- | --- |
| Shortland | Expression | Description |
| $a+=$b | $a=$a+$b | Add two results and assign the result in First |
| $a-=$b | $a=$a-$b | Subtract two number and assign the result in first. |
| $a\*=$b | $a=$a\*$b | Multiply two number and assign the result in first. |
| $a/=$b | $a=$a/$b | Divide two number and assign the result in first |
| $a%=$b | $a=$a%$b | Compute the modulus of two number and assign the result in first. |

**Example:**

<?php

$x1=100;

$x2=200;

$x3=300;

$x4=400;

$x5=500;

$x1+= 100;

echo " $x1 <br />";

$x2-= 200;

echo " $x2 <br />";

$x3\*= 300;

echo " $x3 <br />";

$x4/= 400;

echo " $x4 <br />";

$x5%= 500;

echo " $x5 <br />";

?>

**Output:**

200

0

90000

1

0

* **PHP Comparison Operator**

In Php Comparison Operator take place values (number and string ) as arguments and evaluate to either true or false.

List of Comaprison Operator

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Example | Result |
| == | Equal | $a==$b | True if $a is exactly equal to $b |
| === | Identical | $a===$b | True if $a is exactly equal to $b and they have same type. |
| != | Not Equal | $a!=$b | True if $a is exactly not equal to $b |
| !== | Not Identical | $a!==$b | True if $a is exactly not equal to $b and they are not of same type. |
| < | Less than | $a<$b | True , if $a is less than $b |
| <= | Less than or equal to | $a<=$b | True, if $a is less than or equal to $b |
| > | Greater than | $a>$b | True, if $a is greater than $b |
| >= | Greater than or equal to | $a>=$b | True, if $a is greater than $b |

**Example(==)Operator**

<?php

$x = 300;

$y = "300";

var\_dump ($x == $y);

?>

**Output:** bool(True)

php codes return true though the type of $x and $y are not equal (the first one is integer type and the second one is character type) but their values are equal.

**Example(===)Operator**

<?php

$x = 300;

$y = "300";

var\_dump($x === $y);

?>

**Output:**bool(False)

php codes returns false as the strict equal operator will compare both value and type of $x and $y.

**Example(!=)Operator**

<?php

$x = 150;

$y = "150";

var\_dump ($x != $y);

?>

**Output:** bool(False)

php codes return false though the type of $x and $y are not equal (the first one is integer type and the second one is character type) but their values are equal.

**Example(!==)Operator**

<?php

$x = 150;

$y = "150";

var\_dump($x !== $y);

?>

**Output:**bool(True)

php codes return true though their values are equal but the type of $x and $y are not equal (the first one is integer type and the second one is character type).

**Example: (<)Operator**

<?php

$x = 100;

$y = 300;

var\_dump($x<$y);

?>

**Output :**

bool(true)

php codes return true as the value of $x is less than $y.

**Example: (<=)Operator**

<?php

$x = 300;

$y = 100;

var\_dump($x<=$y);

?>

Output :

bool(false)

php codes return true as the value of $x is less than $y.

**Example: (>)Operator**

<?php

$x = 100;

$y = 300;

var\_dump($x>$y);

?>

Output :

bool(false)

php codes return false as the value of $x is not greater than $y.

**Example: (>=)Operator**

<?php

$x = 100;

$y = 300;

var\_dump($x>=$y);

?>

Output :

bool(false)

php codes return false as the value of $x is not greater than or equal to $y.

**PHP: Incrementing Decrementing Operators**

PHP supports C-style pre and post increment and decrement operators. The Increment/decrement operators operate only on variables and not on any value.

|  |  |  |
| --- | --- | --- |
| Example | Name | Effect |
| ++a | Pre-increment | Increment $a by 1, then return $x |
| a++ | Post-increment | Return $a, then increment by1 |
| --a | Pre-decrement | Decrement $a by 1, then return $a |
| a-- | Post-decrement | Return $a, then decrement by 1 |

<?php

$a = 10;

echo 'Value of $a is :'.$a;

echo '<br />After Pre-increment value of $a ( i.e. ++$a ) is: '.++$a;

$a = 20;

echo '<br />Value of $a is :'.$a;

echo '<br />After Post-increment value of $a ( i.e. $a++ ) is: '.$a++;

$a = 30;

echo '<br />Value of $a is :'.$a;

echo '<br />After Pre-decrement value of $a ( i.e. --$a ) is: '.--$a;

$a = 40;

echo '<br />Value of $a is :'.$a;

echo '<br />After Post-decrement value of $a ( i.e. $a-- ) is: '.$a--;

?>

**Output:**

Value of $a is :10

After Pre-increment value of $a ( i.e. ++$a ) is: 11

Value of $a is :20

After Post-increment value of $a ( i.e. $a++ ) is: 20

Value of $a is :30

After Pre-decrement value of $a ( i.e. --$a ) is: 29

Value of $a is :40

After Post-decrement value of $a ( i.e. $a-- ) is: 40**PHP : Logical Operators**

The standard logical operators and, or, not, and xor are supported by PHP. Logical operators first convert their operands to boolean values and then perform the respective comparison.

List of logical operators:

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Example | Result |
| && | AND | $x && $y | is true if both $x and $y are true. |
| || | OR | $x || $y | is true if either $x or $y is true. |
| ! | NOT | !$x | is true if $x is not true. |

**PHP String operator**

There are two string operators : concatenation operator ('.') and concatenating assignment operator ('.=').

<?php

$x = "Good";

$y = $x ." Morning";

// now $y contains "Good Morning "

echo $y;

?>

OUTPUT:

Good Morning

Example: PHP string concatenating assignment operator

<?php

$x = "Good";

$x.= " Morning";

echo $x;

?>

**DECISION MAKING/CONDITIONAL STATEMENT**

* PHP If Statement

PHP if statement is executed if condition is true.

Syntax:

if(condition)

{

//code to be executed

}

Program:

<?php

$a=10;

If($a<20)

{

Echo “ a is less than 20”;

}

?>

**PHP If-else Statement**

PHP if-else statement is executed whether condition is true or false.

**Syntax:**

if(condition){

//code to be executed if true

}else{

//code to be executed if false

}

**Example:**

<?php

$number=12;

if($number%2==0)

{

echo "$number is even";

}

Else

{

echo "$number is odd";

}

?>

**OUTPUT:** Number is even.

**PHP Switch**

PHP switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement.

**Syntax**

switch(expression){

case value1:

//code to be executed

break;

case value2:

//code to be executed

break; ......

default:

code to be executed if all cases are not matched;

}

**Program:**

<?php

$num=20;

switch($num)

{

case 10:

echo("number is equals to 10");

break;

case 20:

echo("number is equal to 20");

break;

case 30:

echo("number is equal to 30");

break;

default:

echo("number is not equal to 10, 20 or 30");

}

?>

**Output:** number is equal to 20

**PHP While Loop**

The while statement is simple, it executes the statement(s) repeatedly as long as the condition is true. The condition is checked every time at the beginning of the loop.

Initialization, check Condition ,

**Syntax:**

while(condition)

{

//code to be executed

// inc\ decr

}

**Example:**

<?php

$x1=1;

while ($x1<=5)

{

echo "Hello Indore :$x1<br />";

$x1=$x1++;

}

?>

**Output:**

Hello Indore:1

Hello Indore:2

Hello Indore:3

Hello Indore:4

Hello Indore:5

**Do-While loop**

In the case of do while loop the condition is tested after having executed the statements within the loop. This means that do-while would execute its statements at least once, even if the condition fails for the first time itself.

**Syntax:**

do

{

execute the statements;

}

while (condition is true)

**Example:**

<?php

$x1=1;

do {

echo "Hello World:$x1 <br />";

$x1=$x1+1;

}while ($x1<=5)

?>

**Output:**

Hello Indore:1

Hello Indore:2

Hello Indore:3

Hello Indore:4

Hello Indore:5

**Example -2:**

In the following example, the loop will run one time exactly since after the first iteration, when truth expression is checked, it evaluates to FALSE (as $X is not bigger than 5) and the loop execution ends.

<?php

$x1=0;

do {

echo "Increment Number : $x1 <br />";

echo "Hello World <br />";

$x1=$x1+1;

}while ($x1>5)

?>

**Output:**

Increment Number : 0

Hello World

**PHP For Loop**

The PHP for loop allows the user to put all the loop-related statements (i.e. INITIALIZER; CONDITION; INCREMENTOR or DECREMENTOR) in one place.

**Syntax:**

for ( initialize counter ; test counter ; increment counter)

{

execute the statement;

}

initialize counter : Initialize the loop counter value.

test counter : Verify the loop counter whether the condition is true.

increment counter : Increasing the loop counter value.

execute the statement : Execute php statements.

**Example:**

<?php

for ($xint=0; $xint<=4; $xint++)

{

echo "Number is : $xint <br />";

}

?>

**Output:**

Number is : 0

Number is : 1

Number is : 2

Number is : 3

Number is : 4

**PHP Nested For Loop**

We can use for loop inside for loop in PHP, it is known as nested for loop.

<?php

for($i=1;$i<=3;$i++){

for($j=1;$j<=3;$j++){

echo "$i $j<br/>";

}

}

?>

**OUTPUT:**

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

**PHP Break: inside loop**

Let's see a simple example to break the execution of for loop if value of i is equal to 5.

<?php

for($i=1;$i<=10;$i++)

{

echo "$i <br/>";

if($i==5)

{

break;

}

}

?>

**Output:**

1

2

3

4

5

**Continue**

The keyword continue allow us to do this. When the keyword continue executed inside a loop the control automatically passes to the beginning of loop. Continue is usually associated with the if.

**Example:**

In the following example, the list of odd numbers between 1 to 10 have printed. In the while loop we test the remainder (here $x%2) of every number, if the remainder is 0 then it becomes an even number and to avoid printing of even numbers continue statement is immediately used and the control passes to the beginning of the loop.

<?php

$x=1;

echo 'List of odd numbers between 1 to 10 <br />';

while ($x<=10)

{

if (($x % 2)==0)

{

$x++;

continue;

}

else

{

echo $x.'<br />';

$x++;

}

}

?>

**Output:**

List of odd numbers between 1 to 10

1

3

5

7

9

**PHP Functions**

PHP function is a piece of code that can be reused many times. It can take input as argument list and return value. There are thousands of built-in functions in PHP.

**Advantage of PHP Functions**

**Code Reusability:** PHP functions are defined only once and can be invoked many times, like in other programming languages.

**Less Code:** It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.

**Easy to understand:** PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

**PHP User-defined Functions**

We can declare and call user-defined functions easily.

**Syntax:**

function functionname(){

//code to be executed

}

**Example:**

<?php

function myfunction()

{

echo "Good Morning";

}

myfunction();

?>

**Output:**

When we call the above function it will print Good Morning

**Example: Functions within functions**

<?php

function function1()

{

function function2()

{

echo "Good Morning <br>";

}

}

function1();

function2();

?>

**OUTPUT:**

In the above example a function, function1() is declared and another function function2() is declared within function1(). Now execute function1() first and then function2() which print "Good Morning", executing function1() makes funtion2() accessible. Therefore we can not call function2() independently without calling function1().

* **PHP Function Arguments(call by value)**

We can pass the information in PHP function through arguments which is separated by comma.

<?php

function sayHello($name){

echo "Hello $name<br/>";

}

sayHello("riya");

sayHello("jai");

sayHello("mahak");

?>

**Output:**

Hello riya

Hello jai

Hello mahak

* **Example to pass two argument in PHP function. (call by Value)**

<?php

function sayHello($name,$age){

echo "Hello $name, you are $age years old<br/>";

}

sayHello("nisha",27);

sayHello("jai",29);

sayHello("neha",23);

?>

**Output:**

Hello nisha, you are 27 years old

Hello jai, you are 29 years old

Hello neha, you are 23 years old

<?php

function sayHello($name,$age=20)

{

echo "Hello $name Your Age is $age <br/>";

}

sayHello("riya", 22);

sayHello("jai");

sayHello("mahak", 26);

?>

* **Passing arguments by reference (call by reffrernce)**

By default, the function arguments are passed by value. If you want to allow a function to change its arguments, you must pass the arguments by reference. To pass an argument to a function as a reference, simply add an ampersand (&) character before the variable name.

<?php

function cube(&$x)

{

$x = $x \* $x \* $x;

}

$result = 5;

cube($result);

echo $result;

?>

**Output:**

**125**

* **Default arguments values**

In the following example the function wage() use a default parameter. When we have called the function without arguments it takes the default value as an argument.

<?php

function wage($minwage= 100)

{

echo "The wage is : $minwage <br />" ;

}

wage(200);

wage();

wage(100);

?>

**Output:**

The wage is : 200

The wage is : 100

The wage is : 100

**PHP: Returning values**

In PHP values are returned by a return statement. The return values must be specified in the variable. If the statement is called within a function the function is terminated immediately and pass control back to the previous position from which it was called.

The return statement can return any type of data.

**Example:**

<?php

function cube($n){

return $n\*$n\*$n;

}

echo "Cube of 3 is: ".cube(3);

?>

**Output:**

Cube of 3 is: 27

**PHP Arrays**

PHP array is an ordered map (contains value on the basis of key). It is used to hold multiple values of similar type in a single variable.

**Advantage of PHP Array**

* Less Code: We don't need to define multiple variables.
* Easy to traverse: By the help of single loop, we can traverse all the elements of an array.
* Sorting: We can sort the elements of array.
* Continuous memory allocation.

There are 3 types of array in PHP.

* Indexed array — An array with a numeric key.
* Associative array — An array where each key has its own specific value.
* Multidimensional array — An array containing one or more arrays within itself.
* **Numeric Array/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 1:**

<?php

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

echo "Season are: $season[0], $season[1], $season[2] and $season[3]";

?>

**OUTPUT:**

Season are: summer, winter, spring and autumn

**Example 2:**

<?php

$season[0]="summer";

$season[1]="winter";

$season[2]="spring";

$season[3]="autumn";

echo "Season are: $season[0], $season[1], $season[2] and $season[3]";

?>

**Output:**

Season are: summer, winter, spring and autumn

* **PHP Associative Array**

PHP allows you to associate name/label with each array elements in PHP using => symbol.

**Example:**

<?php

$salary=array("Riya"=>"550000","jiya"=>"250000","niraj"=>"200000");

echo "Riya salary: ".$salary["Riya"]."<br/>";

echo "jiya salary: ".$salary["jiya"]."<br/>";

echo "niraj salary: ".$salary["niraj"]."<br/>";

?>

**Output:**

Riya salary: 550000

jiya salary: 250000

niraj salary: 200000

**Example:**

<?php

$salary["Riya"]="550000";

$salary["Jiya"]="250000";

echo "Riya salary: ".$salary["Riya"]."<br/>";

echo "Jiya salary: ".$salary["Jiya"]."<br/>";

?>

**Output:**

Riya salary: 550000

Jiya salary: 250000

* **PHP Multidimensional Array**

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

<?php

// Define nested array

$contacts = array(

array(

"name" => "Peter",

"email" => "peter@mail.com",

),

array(

"name" => "Kent",

"email" => "kent@mail.com",

),

array(

"name" => "Harry",

"email" => "harrypotter@mail.com",

)

);

echo "Peter Email-id is: " . $contacts[0]["email"];

?>

**Output:** Peter Email-id is: [peter@mail.com](mailto:peter@mail.com)

**Viewing Array Structure and Values**

You can see the structure and values of any array by using one of two statements — var\_dump() or print\_r(). The print\_r() statement, however, gives somewhat less information.

**Example:**

<?php

$cities = array("London", "Paris", "New York");

Print\_r($cities);

?>

**Output:**

Array ( [0] => London [1] => Paris [2] => New York )

**Example:**

<?php

$cities = array("London", "Paris", "New York");

var\_dump($cities);

?>

**Output:**

array(3) { [0]=> string(6) "London" [1]=> string(5) "Paris" [2]=> string(8) "New York" }

**PHP Functions For Sorting Arrays**

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.

* **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).

**Example:**

<?php

$colors = array("Red", "Green", "Blue", "Yellow");

sort($colors);

print\_r($colors);

?>

**Output:**

Array ( [0] => Blue [1] => Green [2] => Red [3] => Yellow )

**Example:**

<?php

$numbers = array(1, 2, 2.5, 4, 7, 10);

sort($numbers);

print\_r($numbers);

?>

**Output:**

Array ( [0] => 1 [1] => 2 [2] => 2.5 [3] => 4 [4] => 7 [5] => 10 )

* **Sorting Indexed Arrays in Descending Order**

The rsort() function is used for sorting the elements of the indexed array in descending order.

**Example:**

<?php

$colors = array("Red", "Green", "Blue", "Yellow");

rsort($colors);

print\_r($colors);

?>

**Output:**

Array ( [0] => Yellow [1] => Red [2] => Green [3] => Blue )

**Example:**

<?php

$numbers = array(1, 2, 2.5, 4, 7, 10);

rsort($numbers);

print\_r($numbers);

?>

**Output:**

Array ( [0] => 10 [1] => 7 [2] => 4 [3] => 2.5 [4] => 2 [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("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);

asort($age);

print\_r($age);

?>

**Output:**

Array ( [Harry] => 14 [Peter] => 20 [Clark] => 35 [John] => 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.

**Example:**

<?php

$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);

arsort($age);

print\_r($age);

?>

**Output:** Array ( [John] => 45 [Clark] => 35 [Peter] => 20 [Harry] => 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.

**Example:**

<?php

$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);

ksort($age);

print\_r($age);

?>

**Output:** Array ( [Clark] => 35 [Harry] => 14 [John] => 45 [Peter] => 20 )

* **PHP count() function**

PHP count() function counts all elements in an array.

**Syntax:**

int count ( mixed $array\_or\_countable [, int $mode = COUNT\_NORMAL ] )

**Example:**

<?php

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

echo count($season);

?>

**Output:**

5

* **PHP array\_change\_key\_case() function**

PHP array\_change\_key\_case() function changes the case of all key of an array.

Note: It changes case of key only.

**Syntax:**

array array\_change\_key\_case ( array $array [, int $case = CASE\_LOWER ] )

**Example:**

<?php

$salary=array("Neha"=>"550000","Priya"=>"250000","Raj"=>"200000");

print\_r(array\_change\_key\_case($salary,CASE\_UPPER));

?>

**Output:**

Array ( [NEHA] => 550000 [PRIYA] => 250000 [RAJ] => 200000 )

* **PHP array\_chunk() function**

PHP array\_chunk() function splits array into chunks. By using array\_chunk() method, you can divide array into many parts.

**Syntax:**

array array\_chunk ( array $array , int $size [, bool $preserve\_keys = false ] )

**Example:**

<?php

$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");

print\_r(array\_chunk($salary,2));

?>

**Output:**

Array (

[0] => Array ( [0] => 550000 [1] => 250000 )

[1] => Array ( [0] => 200000 )

)

* **PHP array\_reverse() function**

PHP array\_reverse() function returns an array containing elements in reversed order.

**Syntax:**

array array\_reverse ( array $array [, bool $preserve\_keys = false ] )

**Example**

<?php

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

$reverseseason=array\_reverse($season);

foreach( $reverseseason as $s )

{

echo "$s<br />";

}

?>

**Output:**

autumn

spring

winter

summer

* **PHP array\_search() function**

PHP array\_search() function searches the specified value in an array. It returns key if search is successful.

**Syntax:**

mixed array\_search ( mixed $needle , array $haystack [, bool $strict = false ] )

**Example:**

<?php

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

$key=array\_search("spring",$season);

echo $key;

?>

**Output:**

2

* **PHP array\_intersect() function**

PHP array\_intersect() function returns the intersection of two array. In other words, it returns the matching elements of two array.

**Syntax:**

array array\_intersect ( array $array1 , array $array2 [, array $... ] )

**Example:**

<?php

$name1=array("siya","john","vivek","jiya");

$name2=array("umesh","siya","kartik","jiya");

$name3=array\_intersect($name1,$name2);

foreach( $name3 as $n )

{

echo "$n<br />";

}

?>

**Output:**

siya

jiya

­­­­­­­­­­­­­­­­­­­­­­­­­­­­

**What is the scope of variables?**

In a programming language, the scope of a variable means the areas in the code where the variable is accessible. Our PHP code consists of classes, functions, structure, etc. So, it is important to understand the scope of global and local variables. This concept can play a very important role in professional PHP developers' lives because it requires clear concept building.

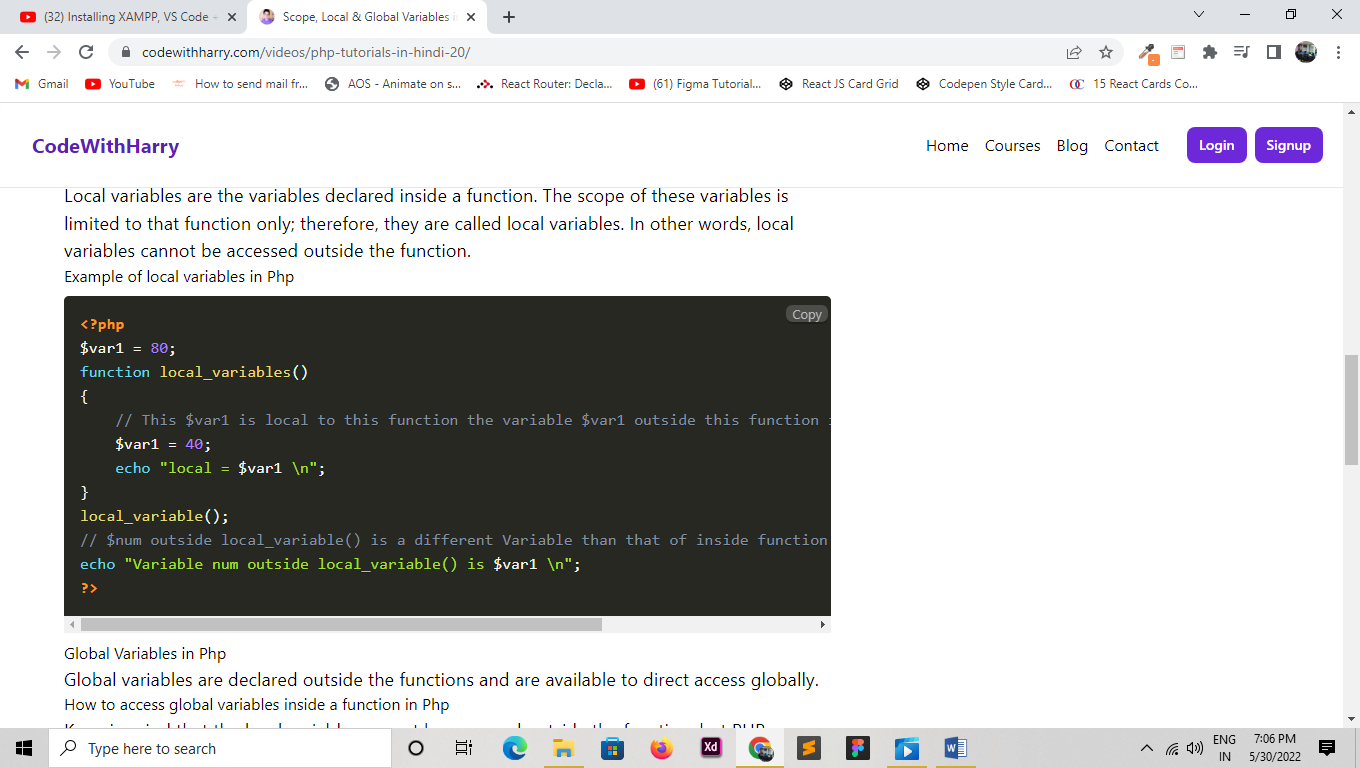
In the next section, we will define local and global variables, and then we will move towards the practical examples for the best outcome out of this PHP lesson.

**What are local and global variables in Php?**

##### Local Variables in Php

Local variables are the variables declared inside a function. The scope of these variables is limited to that function only; therefore, they are called local variables. In other words, local variables cannot be accessed outside the function.

##### **Example of local variables in Php.**

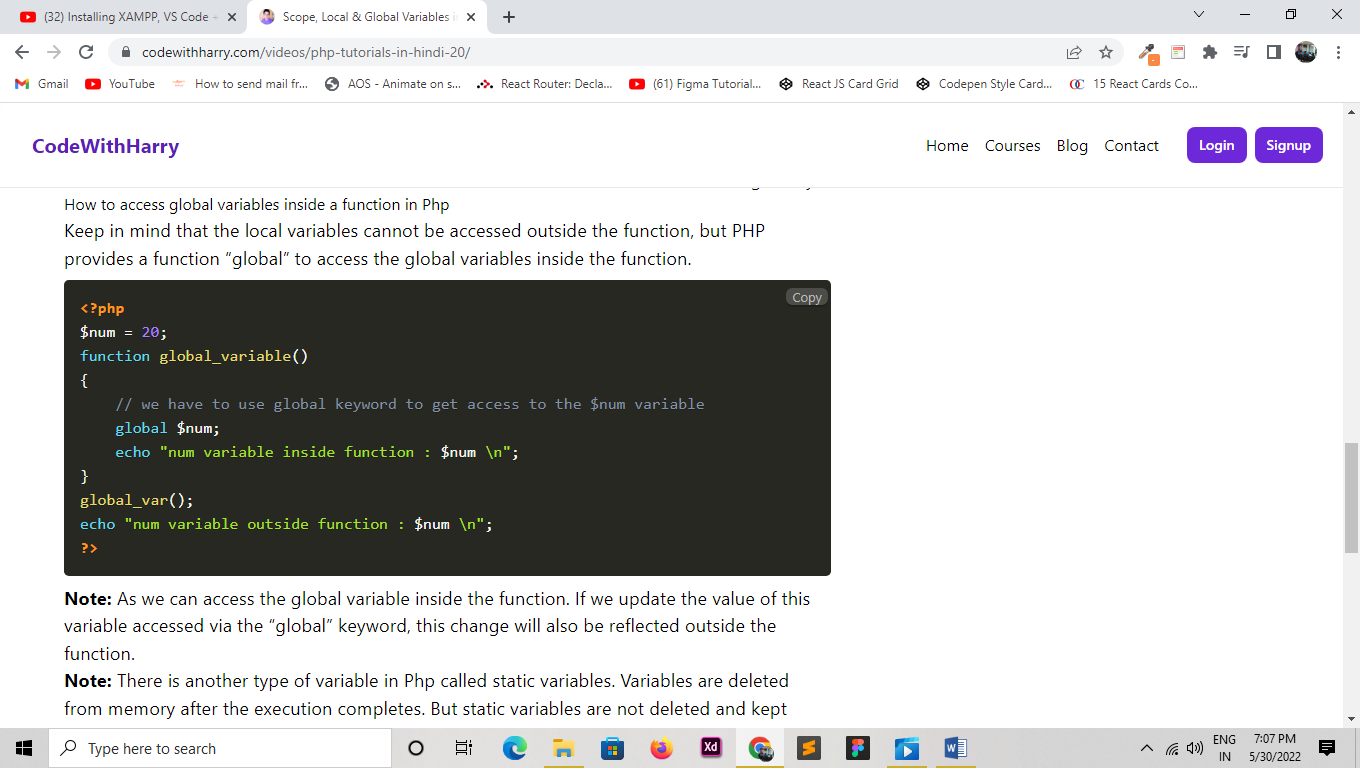


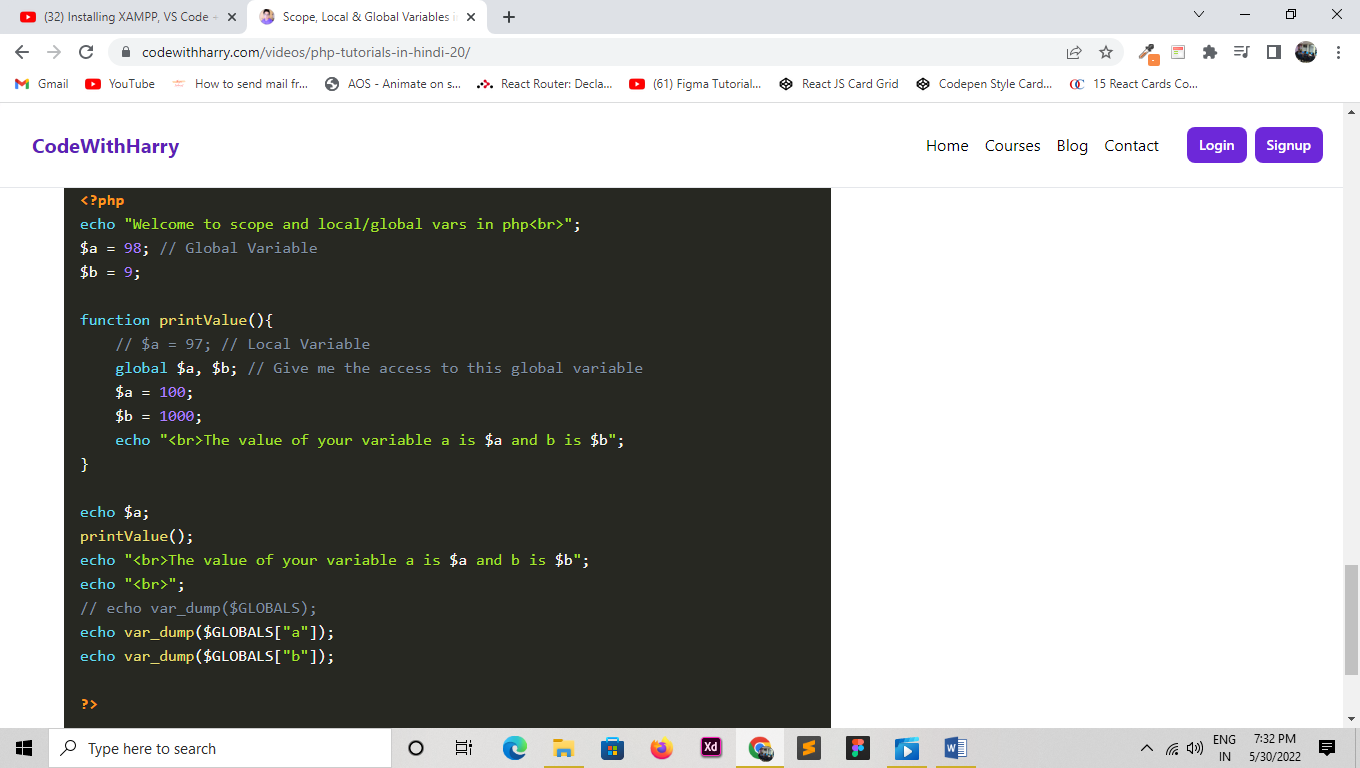
##### **Global Variables in Php**

Global variables are declared outside the functions and are available to direct access globally.

##### How to access global variables inside a function in Php

Keep in mind that the local variables cannot be accessed outside the function, but PHP provides a function “global” to access the global variables inside the function.





**PHP Strings**

A string is a sequence of letters, numbers, special characters and arithmetic values or combination of all.

There are two ways to create a String:

* Single quote
* Double quote

**Single Quoted PHP String**

We can create a string in PHP by enclosing text in a single quote. It is the easiest way to specify string in PHP.

<?php

$str='Hello Indore';

echo $str;

?>

**Output:** Hello Indore within single quote

We can store multiple line text, special characters and escape sequences in a single quoted PHP string.

<?php

$str1='Hello indore

multiple line

text within single quoted string';

$str2='Using double "quote" directly inside single quoted string';

$str3='Using escape sequences \n in single quoted string';

echo "$str1 <br/> $str2 <br/> $str3";

?>

**Output:**

Hello text multiple line text within single quoted string .

Using double "quote" directly inside single quoted string .

Using escape sequences \n in single quoted string.

* **Double Quoted PHP String**

In PHP, we can specify string through enclosing text within double quote also. But escape sequences and variables will be interpreted using double quote PHP strings.

<?php

$str="Hello text within double quote";

echo $str;

?>

**Output:** Hello text within double quote

**Now, you can't use double quote directly inside double quoted string.**

<?php

$str1="Using double "quote" directly inside double quoted string";

echo $str1;

?>

**Output:**

Parse error: syntax error, unexpected

**In double quoted strings, variable will be interpreted.**

<?php

$num=100;

echo "Number is: $num";

?>

**Output:**

Number is: 100

**PHP String Functions**

PHP provides various string functions to access and manipulate strings.

Here Some String functions are:

# **PHP strtolower() function**

The strtolower() function returns string in lowercase letter.

**Syntax:**

string strtolower ( string $string )

**Example-**

<?php

$str="HELLO INDORE";

$str=strtolower($str);

echo $str;

?>

**Output:** hello indore

* **PHP strtoupper() function**

The strtoupper() function returns string in uppercase letter.

**Syntax:**

string strtoupper ( string $string )

**Example:**

<?php

$str="hello everyone";

$str=strtoupper($str);

echo $str;

?>

**Output:**HELLO EVERYONE

* **PHP ucfirst() function**

The ucfirst() function returns string converting first character into uppercase. It doesn't change the case of other characters.

**Syntax:**

string ucfirst ( string $str )

**Example**

<?php

$str="hello indore";

$str=ucfirst($str);

echo $str;

?>

**Output:** Hello indore

* **PHP lcfirst() function**

The lcfirst() function returns string converting first character into lowercase. It doesn't change the case of other characters.

**Syntax:**

string lcfirst ( string $str )

**Example:**

<?php

$str="MY name IS RIYA";

$str=lcfirst($str);

echo $str;

?>

**Output:** mY name IS RIYA

* **PHP ucwords() function**

The ucwords() function returns string converting first character of each word into uppercase.

**Syntax:**

string ucwords ( string $str )

**Example:**

<?php

$str="my name is jiya jaiswal";

$str=ucwords($str);

echo $str;

?>

**Output:**

My Name Is Jiya Jaiswal

* **PHP strrev() function**

The strrev() function returns reversed string.

**Syntax:**

string strrev ( string $string )

**Example:**

<?php

$str="my name is riya jaiswal";

$str=strrev($str);

echo $str;

?>

**Output:**

lawsiaj ayir si eman ym

* **PHP strlen() function**

The strlen() function returns length of the string.

**Syntax:**

int strlen ( string $string )

**Example:**

<?php

$str="my name is Riya jaiswal";

$str=strlen($str);

echo $str;

?>

**Output:** 23

**PHP Form Handling**

We can create and use forms in PHP. To get form data, we need to use PHP superglobals $\_GET and $\_POST.

The form request may be get or post. To retrieve data from get request, we need to use $\_GET, for post request $\_POST.

**PHP Get Form**

Get request is the default form request. The data passed through get request is visible on the URL browser so it is not secured. You can send limited amount of data through get request.

**Example:**

File: form1.html

<form action="new.php" method="get">

Name: <input type="text" name="name"/>

<input type="submit" value="visit"/>

</form>

File: new.php

<?php

$name=$\_GET["name"];//receiving name field value in $name variable

echo "Welcome, $name";

?>

**PHP Post Form**

Post request is widely used to submit form that have large amount of data such as file upload, image upload, login form, registration form etc.

The data passed through post request is not visible on the URL browser so it is secured. You can send large amount of data through post request.

**Example**

**File: form1.html**

<form action="login.php" method="post">

<table>

<tr>

<td>Name:</td>

<td> <input type="text" name="fname"/></td></tr>

<tr>

<td>Password:</td>

<td> <input type="password" name="password"/></td>

</tr>

<tr><td colspan="2"><input type="submit" value="login"/> </td></tr>

</table>

</form>

**File: login.php**

<?php

$name=$\_POST["fname"];

$password=$\_POST["password"];

echo "Welcome: $name, your password is: $password";

?>

**PHP Include File**

PHP allows you to include file so that a page content can be reused many times. There are two ways to include file in PHP.

* include
* require

**Advantage**

Code Reusability: By the help of include and require construct, we can reuse HTML code or PHP script in many PHP scripts.

**PHP include example**

PHP include is used to include file on the basis of given path. You may use relative or absolute path of the file. Let's see a simple PHP include example.

**File: menu.html**

<a href="#">Home</a> |

<a href="#">PHP</a> |

<a href="#">Java</a> |

<a href="#">HTML</a>

**File: include1.php**

<?php include("menu.html"); ?>

<h1>This is Main Page</h1>

* **PHP require example**

PHP require is similar to include. Let's see a simple PHP require example.

**File: menu.html**

<a href=”#">Home</a> |

<a href="#">PHP</a> |

<a href="#">Java</a> |

<a href="#">HTML</a>

**File: require1.php**

<?php require("menu.html"); ?>

<h1>This is Main Page</h1>

**PHP include vs PHP require**

The only difference is — the include() statement will only generate a PHP warning but allow script execution to continue if the file to be included can't be found, whereas the require() statement will generate a fatal error and stops the script execution.

**PHP Date and Time**

* **The PHP Date() Function**

The PHP date() function convert a timestamp to a more readable date and time.

**Syntax**

date(format, timestamp)

The format parameter in the date() function is required which specifies the format of returned date and time. However the timestamp is an optional parameter, if not included then current date and time will be used.

**Example**

<?php

$today = date("d/m/Y");

echo $today;

?>

**Example:**

<?php

echo date("d/m/Y") . "<br>";

echo date("d-m-Y") . "<br>";

echo date("d.m.Y");

?>

* **The PHP time() Function**

The time() function is used to get the current time as a Unix timestamp (the number of seconds since the beginning of the Unix epoch: January 1 1970 00:00:00 GMT).

**Example:**

<?php

// Executed at March 05, 2014 07:19:18

$timestamp = time();

echo($timestamp);

?>

We can convert this timestamp to a human readable date through passing it to the previously introduce date() function.

**Example:**

<?php

$timestamp = 1394003958;

echo(date("F d, Y h:i:s", $timestamp));

?>

**PHP COOKIES**

**What is a Cookie?**

A cookie is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer. They are typically used to keeping track of information such as username that the site can retrieve to personalize the page when user visit the website next time.

**Setting a Cookie in PHP**

The setcookie() function is used to set a cookie in PHP. Make sure you call the setcookie() function before any output generated by your script otherwise cookie will not set. The basic syntax of this function can be given with:

setcookie(name, value, expire, path, domain, secure);

The parameters of the setcookie() function have the following meanings:

Tip:If the expiration time of the cookie is set to 0, or omitted, the cookie will expire at the end of the session i.e. when the browser closes.

Here's an example that uses setcookie() function to create a cookie named username and assign the value value John Carter to it. It also specify that the cookie will expire after 30 days (30 days \* 24 hours \* 60 min \* 60 sec).

**Example**

<?php

// Setting a cookie

setcookie("username", "John Carter", time()+30\*24\*60\*60);

?>

**Note:All the arguments except the name are optional. You may also replace an argument with an empty string ("") in order to skip that argument, however to skip the expire argument use a zero (0) instead, since it is an integer.**

**Accessing Cookies Values**

The PHP $\_COOKIE superglobal variable is used to retrieve a cookie value. It typically an associative array that contains a list of all the cookies values sent by the browser in the current request, keyed by cookie name. The individual cookie value can be accessed using standard array notation, for example to display the username cookie set in the previous example, you could use the following code.

**Example**

<?php

// Accessing an individual cookie value

echo $\_COOKIE["username"];

?>

The PHP code in the above example produce the following output.

John Carter

It's a good practice to check whether a cookie is set or not before accessing its value. To do this you can use the PHP isset() function, like this:

**Example**

<?php

// Verifying whether a cookie is set or not

if(isset($\_COOKIE["username"])){

echo "Hi " . $\_COOKIE["username"];

} else{

echo "Welcome Guest!";

}

?>

You can use the print\_r() function like print\_r($\_COOKIE); to see the structure of this $\_COOKIE associative array, like you with other arrays.

**Removing Cookies**

You can delete a cookie by calling the same setcookie() function with the cookie name and any value (such as an empty string) however this time you need the set the expiration date in the past, as shown in the example below:

**Example**

<?php

// Deleting a cookie

setcookie("username", "", time()-3600);

?>

**PHP SESSION**

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

**What is a PHP Session?**

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

**Starting a PHP Session**

Before you can store any information in session variables, you must first start up the session. To begin a new session, simply call the PHP session\_start() function. It will create a new session and generate a unique session ID for the user.

<?php

// Starting session

session\_start();

?>

The session\_start() function first checks to see if a session already exists by looking for the presence of a session ID. If it finds one, i.e. if the session is already started, it sets up the session variables and if doesn't, it starts a new session by creating a new session ID.

**Storing and Accessing Session Data**

You can store all your session data as key-value pairs in the $\_SESSION[] superglobal array. The stored data can be accessed during lifetime of a session. Consider the following script, which creates a new session and registers two session variables.

ExampleDownload

<?php

// Starting session

session\_start();

// Storing session data

$\_SESSION["firstname"] = "Peter";

$\_SESSION["lastname"] = "Parker";

?>

<?php

// Starting session

session\_start();

// Accessing session data

echo 'Hi, ' . $\_SESSION["firstname"] . ' ' . $\_SESSION["lastname"];

?>

**Note:To access the session data in the same page there is no need to recreate the session since it has been already started on the top of the page.**

**Destroying a Session**

If you want to remove certain session data, simply unset the corresponding key of the $\_SESSION associative array, as shown in the following example:

Example

<?php

// Starting session

session\_start();

// Removing session data

if(isset($\_SESSION["lastname"])){

unset($\_SESSION["lastname"]);

}

?>

However, to destroy a session completely, simply call the session\_destroy() function. This function does not need any argument and a single call destroys all the session data.

**Example**

<?php

// Starting session

session\_start();

// Destroying session

session\_destroy();

?>