//////////////////

<?php  
$color="red";  
echo "My car is " . $color . "<br>";  
echo "My house is " . $COLOR . "<br>";  
echo "My boat is " . $coLOR . "<br>";  
?>

//////////////////////////

Rules for PHP variables:

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case sensitive ($y and $Y are two different variables)

PHP is a Loosely Type Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on its value.

In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

Local and Global Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function.

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function.

The following example tests variables with local and global scope:

Example

<?php  
$x=5; // global scope  
  
function myTest()  
{  
$y=10; // local scope  
echo "<p>Test variables inside the function:<p>";  
echo "Variable x is: $x";  
echo "<br>";  
echo "Variable y is: $y";  
}   
  
myTest();  
  
echo "<p>Test variables outside the function:<p>";  
echo "Variable x is: $x";  
echo "<br>";  
echo "Variable y is: $y";  
?>

PHP The global Keyword

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):

Example

<?php  
$x=5;  
$y=10;  
  
function myTest()  
{  
global $x,$y;  
$y=$x+$y;  
}  
  
myTest();  
echo $y; // outputs 15  
?>

PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

Example

<?php  
$x=5;  
$y=10;  
  
function myTest()  
{  
$GLOBALS['y']=$GLOBALS['x']+$GLOBALS['y'];  
}   
  
myTest();  
echo $y; // outputs 15  
?>

PHP The static Keyword

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:

Example

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

PHP Data Types

« PreviousNext Chapter »

String, Integer, Floating point numbers, Boolean, Array, Object, NULL.

PHP Strings

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

Example

<?php

$x = "Hello world!";

echo $x;

echo "<br>";

$x = 'Hello world!';

echo $x;

?>

Run example »

PHP Integers

An integer is a number without decimals.

Rules for integers:

An integer must have at least one digit (0-9)

An integer cannot contain comma or blanks

An integer must not have a decimal point

An integer can be either positive or negative

Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example we will test different numbers. The PHP var\_dump() function returns the data type and value of variables:

Example

<?php

$x = 5985;

var\_dump($x);

echo "<br>";

$x = -345; // negative number

var\_dump($x);

echo "<br>";

$x = 0x8C; // hexadecimal number

var\_dump($x);

echo "<br>";

$x = 047; // octal number

var\_dump($x);

?>

Run example »

PHP Floating Point Numbers

A floating point number is a number with a decimal point or a number in exponential form.

In the following example we will test different numbers. The PHP var\_dump() function returns the data type and value of variables:

Example

<?php

$x = 10.365;

var\_dump($x);

echo "<br>";

$x = 2.4e3;

var\_dump($x);

echo "<br>";

$x = 8E-5;

var\_dump($x);

?>

Run example »

PHP Booleans

Booleans can be either TRUE or FALSE.

$x=true;

$y=false;

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Arrays

An array stores multiple values in one single variable.

In the following example we create an array, and then use the PHP var\_dump() function to return the data type and value of the array:

Example

<?php

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

var\_dump($cars);

?>

Run example »

You will learn a lot more about arrays in later chapters of this tutorial.

PHP Objects

An object is a data type which stores data and information on how to process that data.

In PHP, an object must be explicitly declared.

First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods.

We then define the data type in the object class, and then we use the data type in instances of that class:

Example

<?php

class Car

{

var $color;

function Car($color="green")

{

$this->color = $color;

}

function what\_color()

{

return $this->color;

}

}

?>

Run example »

You will learn more about objects in a later chapter of this tutorial.

PHP NULL Value

The special NULL value represents that a variable has no value. NULL is the only possible value of data type NULL.

The NULL value identifies whether a variable is empty or not. Also useful to differentiate between the empty string and null values of databases.

Variables can be emptied by setting the value to NULL:

Example

<?php

$x="Hello world!";

$x=null;

var\_dump($x);

?>

PHP Constants

« PreviousNext Chapter »

Constants are like variables except that once they are defined they cannot be changed or undefined.

PHP Constants

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.

Set a PHP Constant

To set a constant, use the define() function - it takes three parameters: The first parameter defines the name of the constant, the second parameter defines the value of the constant, and the optional third parameter specifies whether the constant name should be case-insensitive. Default is false.

The example below creates a case-sensitive constant, with the value of "Welcome to W3Schools.com!":

Example

<?php

define("GREETING", "Welcome to W3Schools.com!");

echo GREETING;

?>

Run example »

The example below creates a case-insensitive constant, with the value of "Welcome to W3Schools.com!":

Example

<?php

define("GREETING", "Welcome to W3Schools.com!", true);

echo greeting;

?>

PHP Operators

« PreviousNext Chapter »

This chapter shows the different operators that can be used in PHP scripts.

PHP Arithmetic Operators

Operator Name Example Result

+ Addition $x + $y Sum of $x and $y

- Subtraction $x - $y Difference of $x and $y

\* Multiplication $x \* $y Product of $x and $y

/ Division $x / $y Quotient of $x and $y

% Modulus $x % $y Remainder of $x divided by $y

The example below shows the different results of using the different arithmetic operators:

Example

<?php

$x=10;

$y=6;

echo ($x + $y); // outputs 16

echo ($x - $y); // outputs 4

echo ($x \* $y); // outputs 60

echo ($x / $y); // outputs 1.6666666666667

echo ($x % $y); // outputs 4

?>

Run example »

PHP Assignment Operators

The PHP assignment operators is used to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment Same as... Description

x = y x = y The left operand gets set to the value of the expression on the right

x += y x = x + y Addition

x -= y x = x - y Subtraction

x \*= y x = x \* y Multiplication

x /= y x = x / y Division

x %= y x = x % y Modulus

The example below shows the different results of using the different assignment operators:

Example

<?php

$x=10;

echo $x; // outputs 10

$y=20;

$y += 100;

echo $y; // outputs 120

$z=50;

$z -= 25;

echo $z; // outputs 25

$i=5;

$i \*= 6;

echo $i; // outputs 30

$j=10;

$j /= 5;

echo $j; // outputs 2

$k=15;

$k %= 4;

echo $k; // outputs 3

?>

Run example »

PHP String Operators

Operator Name Example Result

. Concatenation $txt1 = "Hello"

$txt2 = $txt1 . " world!" Now $txt2 contains "Hello world!"

.= Concatenation assignment $txt1 = "Hello"

$txt1 .= " world!" Now $txt1 contains "Hello world!"

The example below shows the results of using the string operators:

Example

<?php

$a = "Hello";

$b = $a . " world!";

echo $b; // outputs Hello world!

$x="Hello";

$x .= " world!";

echo $x; // outputs Hello world!

?>

Run example »

PHP Increment / Decrement Operators

Operator Name Description

++$x Pre-increment Increments $x by one, then returns $x

$x++ Post-increment Returns $x, then increments $x by one

--$x Pre-decrement Decrements $x by one, then returns $x

$x-- Post-decrement Returns $x, then decrements $x by one

The example below shows the different results of using the different increment/decrement operators:

Example

<?php

$x=10;

echo ++$x; // outputs 11

$y=10;

echo $y++; // outputs 10

$z=5;

echo --$z; // outputs 4

$i=5;

echo $i--; // outputs 5

?>

Run example »

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator Name Example Result

== Equal $x == $y True if $x is equal to $y

=== Identical $x === $y True if $x is equal to $y, and they are of the same type

!= Not equal $x != $y True if $x is not equal to $y

<> Not equal $x <> $y True if $x is not equal to $y

!== Not identical $x !== $y True if $x is not equal to $y, or they are not of the same type

> Greater than $x > $y True if $x is greater than $y

< Less than $x < $y True if $x is less than $y

>= Greater than or equal to $x >= $y True if $x is greater than or equal to $y

<= Less than or equal to $x <= $y True if $x is less than or equal to $y

The example below shows the different results of using some of the comparison operators:

Example

<?php

$x=100;

$y="100";

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

echo "<br>";

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

echo "<br>";

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

echo "<br>";

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

echo "<br>";

$a=50;

$b=90;

var\_dump($a > $b);

echo "<br>";

var\_dump($a < $b);

?>

Run example »

PHP Logical Operators

Operator Name Example Result

and And $x and $y True if both $x and $y are true

or Or $x or $y True if either $x or $y is true

xor Xor $x xor $y True if either $x or $y is true, but not both

&& And $x && $y True if both $x and $y are true

|| Or $x || $y True if either $x or $y is true

! Not !$x True if $x is not true

PHP Array Operators

The PHP array operators are used to compare arrays:

Operator Name Example Result

+ Union $x + $y Union of $x and $y (but duplicate keys are not overwritten)

== Equality $x == $y True if $x and $y have the same key/value pairs

=== Identity $x === $y True if $x and $y have the same key/value pairs in the same order and of the same types

!= Inequality $x != $y True if $x is not equal to $y

<> Inequality $x <> $y True if $x is not equal to $y

!== Non-identity $x !== $y True if $x is not identical to $y

The example below shows the different results of using the different array operators:

Example

<?php

$x = array("a" => "red", "b" => "green");

$y = array("c" => "blue", "d" => "yellow");

$z = $x + $y; // union of $x and $y

var\_dump($z);

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

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

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

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

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

?>

PHP if...else...elseif Statements

« PreviousNext Chapter »

Conditional statements are used to perform different actions based on different conditions.

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

if statement - executes some code only if a specified condition is true

if...else statement - executes some code if a condition is true and another code if the condition is false

if...elseif....else statement - selects one of several blocks of code to be executed

switch statement - selects one of many blocks of code to be executed

PHP - The if Statement

The if statement is used to execute some code only if a specified condition is true.

Syntax

if (condition)

{

code to be executed if condition is true;

}

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

<?php

$t=date("H");

if ($t<"20")

{

echo "Have a good day!";

}

?>

Run example »

PHP - The if...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is false.

Syntax

if (condition)

{

code to be executed if condition is true;

}

else

{

code to be executed if condition is false;

}

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

<?php

$t=date("H");

if ($t<"20")

{

echo "Have a good day!";

}

else

{

echo "Have a good night!";

}

?>

Run example »

PHP - The if...elseif....else Statement

Use the if....elseif...else statement to select one of several blocks of code to be executed.

Syntax

if (condition)

{

code to be executed if condition is true;

}

elseif (condition)

{

code to be executed if condition is true;

}

else

{

code to be executed if condition is false;

}

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

<?php

$t=date("H");

if ($t<"10")

{

echo "Have a good morning!";

}

elseif ($t<"20")

{

echo "Have a good day!";

}

else

{

echo "Have a good night!";

}

?>

PHP switch Statement

« PreviousNext Chapter »

The switch statement is used to perform different actions based on different conditions.

The PHP switch Statement

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

Syntax

switch (n)

{

case label1:

code to be executed if n=label1;

break;

case label2:

code to be executed if n=label2;

break;

case label3:

code to be executed if n=label3;

break;

...

default:

code to be executed if n is different from all labels;

}

This is how it works: First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically. The default statement is used if no match is found.

Example

<?php

$favcolor="red";

switch ($favcolor)

{

case "red":

echo "Your favorite color is red!";

break;

case "blue":

echo "Your favorite color is blue!";

break;

case "green":

echo "Your favorite color is green!";

break;

default:

echo "Your favorite color is neither red, blue, or green!";

}

?>

PHP while Loops

« PreviousNext Chapter »

PHP while loops execute a block of code while the specified condition is true.

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

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

The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

while (condition is true)

{

code to be executed;

}

The example below first sets a variable $x to 1 ($x=1;). Then, the while loop will continue to run as long as $x is less than, or equal to 5. $x will increase by 1 each time the loop runs ($x++;):

Example

<?php

$x=1;

while($x<=5)

{

echo "The number is: $x <br>";

$x++;

}

?>

Run example »

The PHP do...while Loop

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

Syntax

do

{

code to be executed;

}

while (condition is true);

The example below first sets a variable $x to 1 ($x=1;). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

<?php

$x=1;

do

{

echo "The number is: $x <br>";

$x++;

}

while ($x<=5)

?>

Run example »

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition fails the first time.

The example below sets the $x variable to 6, then it runs the loop, and then the condition is checked:

Example

<?php

$x=6;

do

{

echo "The number is: $x <br>";

$x++;

}

while ($x<=5)

?>

PHP for Loops

« PreviousNext Chapter »

PHP for loops execute a block of code a specified number of times.

The PHP for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

for (init counter; test counter; increment counter)

{

code to be executed;

}

Parameters:

init counter: Initialize the loop counter value

test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.

increment counter: Increases the loop counter value

The example below displays the numbers from 0 to 10:

Example

<?php

for ($x=0; $x<=10; $x++)

{

echo "The number is: $x <br>";

}

?>

Run example »

The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

foreach ($array as $value)

{

code to be executed;

}

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array ($colors):

Example

<?php

$colors = array("red","green","blue","yellow");

foreach ($colors as $value)

{

echo "$value <br>";

}

?>

PHP Functions

« PreviousNext Chapter »

The real power of PHP comes from its functions; it has more than 1000 built-in functions.

PHP 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 in PHP

A user defined function declaration starts with the word "function":

Syntax

function functionName()

{

code to be executed;

}

Note: A function name can start with a letter or underscore (not a number).

Tip: Give the function a name that reflects what the function does!

Note Remember that function names are case-insensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ( { ) indicates the beginning of the function code and the closing curly brace ( } ) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

<?php

function writeMsg()

{

echo "Hello world!";

}

writeMsg(); // call the function

?>

Run example »

PHP Function Arguments

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. You can add as many arguments as you want, just seperate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

<?php

function familyName($fname)

{

echo "$fname Refsnes.<br>";

}

familyName("Jani");

familyName("Hege");

familyName("Stale");

familyName("Kai Jim");

familyName("Borge");

?>

Run example »

The following example has a function with two arguments ($fname and $year):

Example

<?php

function familyName($fname,$year)

{

echo "$fname Refsnes. Born in $year <br>";

}

familyName("Hege","1975");

familyName("St�le","1978");

familyName("Kai Jim","1983");

?>

Run example »

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

<?php

function setHeight($minheight=50)

{

echo "The height is : $minheight <br>";

}

setHeight(350);

setHeight(); // will use the default value of 50

setHeight(135);

setHeight(80);

?>

Run example »

PHP Functions - Returning values

To let a function return a value, use the return statement:

Example

<?php

function sum($x,$y)

{

$z=$x+$y;

return $z;

}

echo "5 + 10 = " . sum(5,10) . "<br>";

echo "7 + 13 = " . sum(7,13) . "<br>";

echo "2 + 4 = " . sum(2,4);

?>

PHP Arrays

« PreviousNext Chapter »

An array stores multiple values in one single variable:

Example

<?php

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

echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";

?>

Run example »

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

$cars1="Volvo";

$cars2="BMW";

$cars3="Toyota";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array:

array();

In PHP, there are three types of arrays:

Indexed arrays - Arrays with numeric index

Associative arrays - Arrays with named keys

Multidimensional arrays - Arrays containing one or more arrays

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0):

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

or the index can be assigned manually:

$cars[0]="Volvo";

$cars[1]="BMW";

$cars[2]="Toyota";

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

<?php

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

echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";

?>

Run example »

Get The Length of an Array - The count() Function

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

Example

<?php

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

echo count($cars);

?>

Run example »

Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a for loop, like this:

Example

<?php

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

$arrlength=count($cars);

for($x=0;$x<$arrlength;$x++)

{

echo $cars[$x];

echo "<br>";

}

?>

Run example »

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

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

or:

$age['Peter']="35";

$age['Ben']="37";

$age['Joe']="43";

The named keys can then be used in a script:

Example

<?php

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

echo "Peter is " . $age['Peter'] . " years old.";

?>

Run example »

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

Example

<?php

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

foreach($age as $x=>$x\_value)

{

echo "Key=" . $x . ", Value=" . $x\_value;

echo "<br>";

}

?>

PHP Sorting Arrays

« PreviousNext Chapter »

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

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

Sort Array in Ascending Order - sort()

The following example sorts the elements of the $cars array in ascending alphabetical order:

Example

<?php

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

sort($cars);

?>

Run example »

The following example sorts the elements of the $numbers array in ascending numerical order:

Example

<?php

$numbers=array(4,6,2,22,11);

sort($numbers);

?>

Run example »

Sort Array in Descending Order - rsort()

The following example sorts the elements of the $cars array in descending alphabetical order:

Example

<?php

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

rsort($cars);

?>

Run example »

The following example sorts the elements of the $numbers array in descending numerical order:

Example

<?php

$numbers=array(4,6,2,22,11);

rsort($numbers);

?>

Run example »

Sort Array in Ascending Order, According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

<?php

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

asort($age);

?>

Run example »

Sort Array in Ascending Order, According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

<?php

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

ksort($age);

?>

Run example »

Sort Array in Descending Order, According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

<?php

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

arsort($age);

?>

Run example »

Sort Array in Descending Order, According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

<?php

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

krsort($age);

?>

PHP Global Variables - Superglobals

« PreviousNext Chapter »

Superglobals were introduced in PHP 4.1.0, and are built-in variables that are always available in all scopes.

PHP Global Variables - Superglobals

Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

$GLOBALS

$\_SERVER

$\_REQUEST

$\_POST

$\_GET

$\_FILES

$\_ENV

$\_COOKIE

$\_SESSION

This chapter will explain some of the superglobals, and the rest will be explained in later chapters.

PHP $GLOBAL

$GLOBAL is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called $GLOBALS[index]. The index holds the name of the variable.

The example below shows how to use the super global variable $GLOBAL:

Example

<?php

$x = 75;

$y = 25;

function addition()

{

$GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];

}

addition();

echo $z;

?>

Run example »

In the example above, since z is a variable present within the $GLOBALS array, it is also accessible form outside the function!

PHP $\_SERVER

$\_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

The example below shows how to use some of the elements in $\_SERVER:

Example

<?php

echo $\_SERVER['PHP\_SELF'];

echo "<br>";

echo $\_SERVER['SERVER\_NAME'];

echo "<br>";

echo $\_SERVER['HTTP\_HOST'];

echo "<br>";

echo $\_SERVER['HTTP\_REFERER'];

echo "<br>";

echo $\_SERVER['HTTP\_USER\_AGENT'];

echo "<br>";

echo $\_SERVER['SCRIPT\_NAME'];

?>

Run example »

The following table lists the most important elements that can go inside $\_SERVER:

Element/Code Description

$\_SERVER['PHP\_SELF'] Returns the filename of the currently executing script

$\_SERVER['GATEWAY\_INTERFACE'] Returns the version of the Common Gateway Interface (CGI) the server is using

$\_SERVER['SERVER\_ADDR'] Returns the IP address of the host server

$\_SERVER['SERVER\_NAME'] Returns the name of the host server (such as www.w3schools.com)

$\_SERVER['SERVER\_SOFTWARE'] Returns the server identification string (such as Apache/2.2.24)

$\_SERVER['SERVER\_PROTOCOL'] Returns the name and revision of the information protocol (such as HTTP/1.1)

$\_SERVER['REQUEST\_METHOD'] Returns the request method used to access the page (such as POST)

$\_SERVER['REQUEST\_TIME'] Returns the timestamp of the start of the request (such as 1377687496)

$\_SERVER['QUERY\_STRING'] Returns the query string if the page is accessed via a query string

$\_SERVER['HTTP\_ACCEPT'] Returns the Accept header from the current request

$\_SERVER['HTTP\_ACCEPT\_CHARSET'] Returns the Accept\_Charset header from the current request (such as utf-8,ISO-8859-1)

$\_SERVER['HTTP\_HOST'] Returns the Host header from the current request

$\_SERVER['HTTP\_REFERER'] Returns the complete URL of the current page (not reliable because not all user-agents support it)

$\_SERVER['HTTPS'] Is the script queried through a secure HTTP protocol

$\_SERVER['REMOTE\_ADDR'] Returns the IP address from where the user is viewing the current page

$\_SERVER['REMOTE\_HOST'] Returns the Host name from where the user is viewing the current page

$\_SERVER['REMOTE\_PORT'] Returns the port being used on the user's machine to communicate with the web server

$\_SERVER['SCRIPT\_FILENAME'] Returns the absolute pathname of the currently executing script

$\_SERVER['SERVER\_ADMIN'] Returns the value given to the SERVER\_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3scholls.com)

$\_SERVER['SERVER\_PORT'] Returns the port on the server machine being used by the web server for communication (such as 80)

$\_SERVER['SERVER\_SIGNATURE'] Returns the server version and virtual host name which are added to server-generated pages

$\_SERVER['PATH\_TRANSLATED'] Returns the file system based path to the current script

$\_SERVER['SCRIPT\_NAME'] Returns the path of the current script

$\_SERVER['SCRIPT\_URI'] Returns the URI of the current page

PHP $\_REQUEST

PHP $\_REQUEST is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $\_REQUEST to collect the value of the input field:

Example

<html>

<body>

<form method="post" action="<?php echo $\_SERVER['PHP\_SELF'];?>">

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

<input type="submit">

</form>

<?php

$name = $\_REQUEST['fname'];

echo $name;

?>

</body>

</html>

Run example »

PHP $\_POST

PHP $\_POST is widely used to collect form data after submitting an HTML form with method="post". $\_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $\_POST to collect the value of the input field:

Example

<html>

<body>

<form method="post" action="<?php echo $\_SERVER['PHP\_SELF'];?>">

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

<input type="submit">

</form>

<?php

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

echo $name;

?>

</body>

</html>

Run example »

PHP $\_GET

PHP $\_GET can also be used to collect form data after submitting an HTML form with method="get".

$\_GET can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

<html>

<body>

<a href="test\_get.php?subject=PHP&web=W3schools.com">Test $GET</a>

</body>

</html>

When a user clicks on the link "Test $GET", the parameters "subject" and "web" is sent to "test\_get.php", and you can then acces their values in "test\_get.php" with $\_GET.

The example below shows the code in "test\_get.php":

Example

<html>

<body>

<?php

echo "Study " . $\_GET['subject'] . " at " . $\_GET['web'];

?>

</body>

</html>

PHP Multidimensional Arrays

« PreviousNext Chapter »

An array can also contain another array as a value, which in turn can hold other arrays as well. In such a way we can create two- or three-dimensional arrays:

Example

<?php

// A two-dimensional array:

$cars = array

(

array("Volvo",100,96),

array("BMW",60,59),

array("Toyota",110,100)

);

?>

Run example »

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

$families = array

(

"Griffin"=>array

(

"Peter",

"Lois",

"Megan"

),

"Quagmire"=>array

(

"Glenn"

),

"Brown"=>array

(

"Cleveland",

"Loretta",

"Junior"

)

);

The array above would look like this if written to the output:

Array

(

[Griffin] => Array

(

[0] => Peter

[1] => Lois

[2] => Megan

)

[Quagmire] => Array

(

[0] => Glenn

)

[Brown] => Array

(

[0] => Cleveland

[1] => Loretta

[2] => Junior

)

)

Example 2

Lets try displaying a single value from the array above:

echo "Is " . $families['Griffin'][2] .

" a part of the Griffin family?";

The code above will output:

Is Megan a part of the Griffin family?