

PHP 7 Tutorial

PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.

PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

Introduction

PHP scripts are executed on the server.

What is PHP?

- 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 is an amazing and popular language!

It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)!

It is deep enough to run the largest social network (Facebook)!

It is also easy enough to be a beginner's first server side language!

What is a 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"

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

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

Installation

What Do I Need?

To start using PHP, you can:

- Find a web host with PHP and MySQL support
- Install a web server on your own PC, and then install PHP and MySQL

Use a Web Host With PHP Support

If your server has activated support for PHP you do not need to do anything.

Just create some .php files, place them in your web directory, and the server will automatically parse them for you.

You do not need to compile anything or install any extra tools.

Because PHP is free, most web hosts offer PHP support.

Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL

The official PHP website (PHP.net) has installation instructions for PHP:

<http://php.net/manual/en/install.php>

Syntax

A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

Basic PHP Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with <?php and ends with ?>:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is ".php".

A PHP file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

Note: PHP statements end with a semicolon (;).

Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand what you are doing
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports several ways of commenting:

Example

```
<!DOCTYPE html>
<html>
<body>
```

```

<?php
// This is a single-line comment

# This is also a single-line comment

/*
This is a multiple-lines comment block
that spans over multiple
lines
*/

// You can also use comments to leave out parts of a code line
$x = 5 /* + 15 */ + 5;
echo $x;
?>

</body>
</html>

```

PHP Case Sensitivity

In PHP, NO keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are case-sensitive.

In the example below, all three echo statements below are legal (and equal):

Example

```

<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>

</body>
</html>

```

However; all variable names are case-sensitive.

In the example below, only the first statement will display the value of the \$color variable (this is because \$color, \$COLOR, and \$coLoR are treated as three different variables):

Example

```

<!DOCTYPE html>
<html>
<body>

```

```

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

</body>
</html>

```

Variables

Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the \$ sign, followed by the name of the variable:

Example

```

<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>

```

After the execution of the statements above, the variable \$txt will hold the value Hello world!, the variable \$x will hold the value 5, and the variable \$y will hold the value 10.5.

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Think of variables as containers for storing data.

PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

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 (\$age and \$AGE are two different variables)

Remember: PHP variable names are case-sensitive!

Output Variables

The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:

```
<?php
$txt = "W3Schools.com";
echo "I love $txt!";
?>
```

The following example will produce the same output as the example above:

Example

```
<?php
$txt = "W3Schools.com";
echo "I love " . $txt . "!";
?>
```

The following example will output the sum of two variables:

Example

```
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
```

PHP is a Loosely Typed 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.

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope

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

Example

```
<?php
$x = 5; // global scope

function myTest() {
    // using x inside this function will generate an error
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

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

Example

```
<?php
function myTest() {
    $x = 5; // local scope
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

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();
?>
```

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

Note: The variable is still local to the function.

echo and print Statements

In PHP there are two basic ways to get output: echo and print.

In this tutorial we use echo (and print) in almost every example. So, this chapter contains a little more info about those two output statements.

PHP echo and print Statements

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.

The PHP echo Statement

The echo statement can be used with or without parentheses: echo or echo().

Display Text

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

```
<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>
```

Display Variables

The following example shows how to output text and variables with the echo statement:

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>
```

The PHP print Statement

The print statement can be used with or without parentheses: print or print().

Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

```
<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
```

Display Variables

The following example shows how to output text and variables with the print statement:

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

print "<h2>" . $txt1 . "</h2>";
print "Study PHP at " . $txt2 . "<br>";
print $x + $y;
?>
```

Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL

PHP String

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!";
$y = 'Hello world!';
```

```
echo $x;
echo "<br>";
echo $y;
?>
```

Resource

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- 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 \$x is an integer. The PHP var_dump() function returns the data type and value:
-

Example

```
<?php
$x = 5985;
var_dump($x);
?>
```

PHP Float

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

In the following example \$x is a float. The PHP var_dump() function returns the data type and value:

```
<?php
$x = 10.365;
var_dump($x);
?>
```

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
```

```
$y = false;
```

Booleans are often used in conditional testing.

PHP Array

An array stores multiple values in one single variable.

In the following example \$cars is an array. The PHP var_dump() function returns the data type and value:

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

PHP Object

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:

Example

```
<?php
class Car {
    function Car() {
        $this->model = "VW";
    }
}
```

```
// create an object
$herbie = new Car();
```

```
// show object properties
echo $herbie->model;
?>
```

PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

Tip: If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:

Example

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.

A common example of using the resource data type is a database call.

Strings

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

PHP String Functions

Get The Length of a String

The PHP **strlen()** function returns the length of a string.

The example below returns the length of the string "Hello world!":

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

Count The Number of Words in a String

The PHP **str_word_count()** function counts the number of words in a string:

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

Reverse a String

The PHP **strrev()** function reverses a string:

Example

```
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

Search For a Specific Text Within a String

The PHP **strpos()** function searches for a specific text within a string.

If a match is found, the function returns the character position of the first match.

If no match is found, it will return FALSE.

The example below searches for the text "world" in the string "Hello world!":

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

Tip: The first character position in a string is 0 (not 1).

Replace Text Within a String

The PHP **str_replace()** function replaces some characters with some other characters in a string.

The example below replaces the text "world" with "Dolly":

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!
?>
```

Constants

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.

Create a PHP Constant

To create a constant, use the `define()` function.

Syntax

`define(name, value, case-insensitive)`

Parameters:

- **name:** Specifies the name of the constant
- **value:** Specifies the value of the constant
- **case-insensitive:** Specifies whether the constant name should be case-insensitive. Default is false

The example below creates a constant with a case-sensitive name:

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

The example below creates a constant with a case-insensitive name:

```
<?php
define("GREETING", "Welcome to W3Schools.com!", true);
echo greeting;
?>
```

PHP7 Constant Arrays

In PHP7, you can create a Array constant using the `define()` function.

The example below creates an Array constant:

```
<?php
define("cars", [
    "Alfa Romeo",
    "BMW",
    "Toyota"
]);
echo cars[0];
?>
```

Constants are Global

Constants are automatically global and can be used across the entire script.

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

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

continues...

if...else...elseif Statements

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 conditions. You can use conditional statements in your code to do this.

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
-

PHP - The if Statement

The if statement executes some code if one 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!";
}
?>
```

PHP - The if...else Statement

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;
}
```

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!";
}
?>
```

PHP - The 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;
}
```


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!";
}
?>
```

switch Statement

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

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, nor green!";
}
?>
```

while Loops

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 <= 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++;
}
?>
```

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);
?>
```

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 is false 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);  
?>
```

for Loops

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

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;  
}
```

The example below displays the numbers from 0 to 10:

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

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):

```
<?php  
$colors = array("red", "green", "blue", "yellow");  
  
foreach ($colors as $value) {  
    echo "$value <br>";  
}  
?>
```

Functions

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!

Function names are NOT case-sensitive.

Example

```
<?php  
function writeMsg() {  
    echo "Hello world!";  
}
```

```
writeMsg(); // call the function  
?>
```

PHP is a Loosely Typed Language

In PHP 7, type declarations were added. This gives us an option to specify the data type expected when declaring a function, and by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

In the following example we try to add a number and a string with without the strict requirement:

```
<?php  
function addNumbers(int $a, int $b) {  
    return $a + $b;  
}  
echo addNumbers(5, "5 days");  
// since strict is NOT enabled "5 days" is changed to int(5), and it will return 10  
?>
```

In the following example we try to add a number and a string with the strict requirement:

```
<?php declare(strict_types=1); // strict requirement

function addNumbers(int $a, int $b) {
    return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is enabled and "5 days" is not an integer, an error will be thrown
?>
```

To specify strict we need to set **declare(strict_types=1);**. This must be the on the very first line of the PHP file. Declaring strict specifies that function calls made in that file must strictly adhere to the specified data types

The strict declaration can make code easier to read, and it forces things to be used in the intended way.

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:

```
<?php declare(strict_types=1); // strict requirement
function setHeight(int $minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

PHP Functions - Returning values

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

Example

```
<?php declare(strict_types=1); // strict requirement
function sum(int $x, int $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 Return Type Declarations

PHP 7 also supports Type Declarations for the return statement. Like with the type declaration for function arguments, by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

To declare a type for the function return, **add a colon (:) and the type right before the opening curly ({)bracket** when declaring the function.

In the following example we specify the return type for the function:

Example

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : float {
    return $a + $b;
}
echo addNumbers(1.2, 5.2);
?>
```

You can specify a different return type, than the argument types, but make sure the return is the correct type:

Example

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : int {
    return (int)($a + $b);
}
echo addNumbers(1.2, 5.2);
?>
```

Arrays

An array stores multiple values in one single variable:

PHP Indexed Arrays

There are two ways to create indexed arrays:

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

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

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] . ".";
?>
```

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 a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

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);
?>
```

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");
$arlength = count($cars);

for($x = 0; $x < $arlength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
?>
```

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>";
}
?>
```

Sorting Arrays

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:

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

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

```
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
?>
```

Sort Array in Descending Order - `rsort()`

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

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

```
rsort($cars);  
?>
```

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

```
<?php  
$numbers = array(4, 6, 2, 22, 11);  
rsort($numbers);  
?>
```

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

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

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

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

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

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

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

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

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

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

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

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

Global Variables - Superglobals

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
```

PHP \$GLOBALS

\$GLOBALS 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 \$GLOBALS:

```
<?php  
$x = 75;  
$y = 25;  
  
function addition() {  
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];  
}  
  
addition();  
echo $z;  
?>
```

In the example above, since z is a variable present within the \$GLOBALS array, it is also accessible from 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'];
?>
```

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

Element/Code	Description
<code>\$_SERVER['PHP_SELF']</code>	Returns the filename of the currently executing script
<code>\$_SERVER['GATEWAY_INTERFACE']</code>	Returns the version of the Common Gateway Interface (CGI) the server is using
<code>\$_SERVER['SERVER_ADDR']</code>	Returns the IP address of the host server
<code>\$_SERVER['SERVER_NAME']</code>	Returns the name of the host server (such as <code>www.w3schools.com</code>)
<code>\$_SERVER['SERVER_SOFTWARE']</code>	Returns the server identification string (such as <code>Apache/2.2.24</code>)
<code>\$_SERVER['SERVER_PROTOCOL']</code>	Returns the name and revision of the information protocol (such as <code>HTTP/1.1</code>)
<code>\$_SERVER['REQUEST_METHOD']</code>	Returns the request method used to access the page (such as <code>POST</code>)
<code>\$_SERVER['REQUEST_TIME']</code>	Returns the timestamp of the start of the request (such as <code>1377687496</code>)
<code>\$_SERVER['QUERY_STRING']</code>	Returns the query string if the page is accessed via a query string

<code>\$_SERVER['HTTP_ACCEPT']</code>	Returns the Accept header from the current request
<code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code>	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
<code>\$_SERVER['HTTP_HOST']</code>	Returns the Host header from the current request
<code>\$_SERVER['HTTP_REFERER']</code>	Returns the complete URL of the current page (not reliable because not all user-agents support it)
<code>\$_SERVER['HTTPS']</code>	Is the script queried through a secure HTTP protocol
<code>\$_SERVER['REMOTE_ADDR']</code>	Returns the IP address from where the user is viewing the current page
<code>\$_SERVER['REMOTE_HOST']</code>	Returns the Host name from where the user is viewing the current page
<code>\$_SERVER['REMOTE_PORT']</code>	Returns the port being used on the user's machine to communicate with the web server
<code>\$_SERVER['SCRIPT_FILENAME']</code>	Returns the absolute pathname of the currently executing script
<code>\$_SERVER['SERVER_ADMIN']</code>	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@w3schools.com)
<code>\$_SERVER['SERVER_PORT']</code>	Returns the port on the server machine being used by the web server for communication (such as 80)
<code>\$_SERVER['SERVER_SIGNATURE']</code>	Returns the server version and virtual host name which are added to server-generated pages
<code>\$_SERVER['PATH_TRANSLATED']</code>	Returns the file system based path to the current script
<code>\$_SERVER['SCRIPT_NAME']</code>	Returns the path of the current script
<code>\$_SERVER['SCRIPT_URI']</code>	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
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  // collect value of input field
  $name = $_REQUEST['fname'];
  if (empty($name)) {
    echo "Name is empty";
  } else {
    echo $name;
  }
}
?>

</body>
</html>
```

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 the 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
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>

```

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" are sent to "test_get.php", and you can then access their values in "test_get.php" with \$_GET.

The example below shows the code in "test_get.php":

```

<html>
<body>

<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

```

```
</body>
</html>
```

PHP 7 Form

Form Handling

The PHP superglobals `$_GET` and `$_POST` are used to collect form-data.

PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

Example

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "**welcome.php**". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables.

The "welcome.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

The output could be something like this:

Welcome John

Your email address is john.doe@example.com

The same result could also be achieved using the HTTP GET method:

Example

```
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

and "welcome_get.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>

</body>
</html>
```

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

Think SECURITY when processing PHP forms!

This page does not contain any form validation, it just shows how you can send and retrieve form data.

However, the next pages will show how to process PHP forms with security in mind! Proper validation of form data is important to protect your form from hackers and spammers!

GET vs. POST

Both GET and POST create an array (e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as `$_GET` and `$_POST`. These 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.

`$_GET` is an array of variables passed to the current script via the URL parameters.

`$_POST` is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL).

GET also has limits on the amount of information to send. The limitation is about 2000 characters.

However, because the variables are displayed in the URL, it is possible to bookmark the page.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Developers prefer POST for sending form data.

Form Validation

Think SECURITY when processing PHP forms!

These pages will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea.

The HTML code looks like this:

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

E-mail: `<input type="text" name="email">`

Website: `<input type="text" name="website">`

Comment: `<textarea name="comment" rows="5" cols="40"></textarea>`

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

Gender:

`<input type="radio" name="gender" value="female">Female`

`<input type="radio" name="gender" value="male">Male`

`<input type="radio" name="gender" value="other">Other`

The Form Element

The HTML code of the form looks like this:

`<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">`

When the form is submitted, the form data is sent with method="post".

What is the `$_SERVER["PHP_SELF"]` variable?

The `$_SERVER["PHP_SELF"]` is a super global variable that returns the filename of the currently executing script.

So, the `$_SERVER["PHP_SELF"]` sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the `htmlspecialchars()` function?

The `htmlspecialchars()` function converts special characters to HTML entities.

This means that it will replace HTML characters like `<` and `>` with `<` and `>`.

This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Big Note on PHP Form Security

The `$_SERVER["PHP_SELF"]` variable can be used by hackers!

If `PHP_SELF` is used in your page then a user can enter a slash (`/`) and then some Cross Site Scripting (XSS) commands to execute.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like

"http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

So far, so good.

However, consider that a user enters the following URL in the address bar:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

In this case, the above code will be translated to:

```
<form method="post"
action="test_form.php/"><script>alert('hacked')</script>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware of that any JavaScript code can be added inside the <script> tag! A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid \$_SERVER["PHP_SELF"] Exploits?

\$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

```
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post"
action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">
```

The exploit attempt fails, and no harm is done!

Validate Form Data With PHP

The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.

When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

```
<script>location.href('http://www.hacked.com')</script>
```

- this would not be executed, because it would be saved as HTML escaped code, like this:

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
```

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function test_input().

Now, we can check each \$_POST variable with the test_input() function, and the script looks like this:

Example

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = test_input($_POST["name"]);
    $email = test_input($_POST["email"]);
    $website = test_input($_POST["website"]);
    $comment = test_input($_POST["comment"]);
    $gender = test_input($_POST["gender"]);
}

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>
```

Notice: that at the start of the script, we check whether the form has been submitted using `$_SERVER["REQUEST_METHOD"]`. If the `REQUEST_METHOD` is `POST`, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

Forms - Required Fields

This chapter shows how to make input fields required and create error messages if needed.

PHP - Required Fields

From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

In the previous chapter, all input fields were optional.

In the following code we have added some new variables: `$nameErr`, `$emailErr`, `$genderErr`, and `$websiteErr`.

These error variables will hold error messages for the required fields. We have also added an if else statement for each `$_POST` variable. This checks if the `$_POST` variable is empty (with the PHP `empty()` function).

If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the `test_input()` function:

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
```

```

    } else {
        $name = test_input($_POST["name"]);
    }

    if (empty($_POST["email"])) {
        $emailErr = "Email is required";
    } else {
        $email = test_input($_POST["email"]);
    }

    if (empty($_POST["website"])) {
        $website = "";
    } else {
        $website = test_input($_POST["website"]);
    }

    if (empty($_POST["comment"])) {
        $comment = "";
    } else {
        $comment = test_input($_POST["comment"]);
    }

    if (empty($_POST["gender"])) {
        $genderErr = "Gender is required";
    } else {
        $gender = test_input($_POST["gender"]);
    }
}
?>

```

PHP - Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

Example

```

<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";

```

```
$name = $email = $gender = $comment = $website = "";
```

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    if (empty($_POST["name"])) {  
        $nameErr = "Name is required";  
    } else {  
        $name = test_input($_POST["name"]);  
    }  
}
```

```
if (empty($_POST["email"])) {  
    $emailErr = "Email is required";  
} else {  
    $email = test_input($_POST["email"]);  
}
```

```
if (empty($_POST["website"])) {  
    $website = "";  
} else {  
    $website = test_input($_POST["website"]);  
}
```

```
if (empty($_POST["comment"])) {  
    $comment = "";  
} else {  
    $comment = test_input($_POST["comment"]);  
}
```

```
if (empty($_POST["gender"])) {  
    $genderErr = "Gender is required";  
} else {  
    $gender = test_input($_POST["gender"]);  
}  
}
```

```
function test_input($data) {  
    $data = trim($data);  
    $data = stripslashes($data);  
    $data = htmlspecialchars($data);  
    return $data;  
}  
?>
```

<h2>PHP Form Validation Example</h2>

<p>* required field</p>

<form method="post" action="<?php echo
htmlspecialchars(\$_SERVER["PHP_SELF"]);?>">

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

* <?php echo \$nameErr;?>


```

E-mail: <input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Website: <input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>

```

Forms - Validate E-mail and URL

This chapter shows how to validate names, e-mails, and URLs.

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```

$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
    $nameErr = "Only letters and white space allowed";
}

```

The `preg_match()` function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's **filter_var() function**.

In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $emailErr = "Invalid email format";
}
```

PHP - Validate URL

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if
(!preg_match("/\b(?:(:|https?|ftp):\/\/|www\.|)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%?=~_|]/i",$website)) {
    $websiteErr = "Invalid URL";
}
```

PHP - Validate Name, E-mail, and URL

Now, the script looks like this:

Example

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    } else {
        $name = test_input($_POST["name"]);
        // check if name only contains letters and whitespace
        if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
            $nameErr = "Only letters and white space allowed";
        }
    }
}

if (empty($_POST["email"])) {
    $emailErr = "Email is required";
} else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
```

```

        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $emailErr = "Invalid email format";
        }
    }

    if (empty($_POST["website"])) {
        $website = "";
    } else {
        $website = test_input($_POST["website"]);
        // check if URL address syntax is valid (this regular expression also allows
        dashes in the URL)
        if
        (!preg_match("/\b(?:(:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=\~_|!:,.;]*[-
a-z0-9+&@#\/%=\~_|/i",$website)) {
            $websiteErr = "Invalid URL";
        }
    }

    if (empty($_POST["comment"])) {
        $comment = "";
    } else {
        $comment = test_input($_POST["comment"]);
    }

    if (empty($_POST["gender"])) {
        $genderErr = "Gender is required";
    } else {
        $gender = test_input($_POST["gender"]);
    }
}
?>

```

Complete Form Example

This chapter shows how to keep the values in the input fields when the user hits the submit button.

PHP - Keep The Values in The Form

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website.

In the comment textarea field, we put the script between the <textarea> and </textarea> tags. The little script outputs the value of the \$name, \$email, \$website, and \$comment variables.

Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

Name: <input type="text" name="name" value="<?php echo \$name;?>">

E-mail: <input type="text" name="email" value="<?php echo \$email;?>">

Website: <input type="text" name="website" value="<?php echo \$website;?>">

Comment: <textarea name="comment" rows="5" cols="40"><?php echo \$comment;?></textarea>

Gender:

```
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="other") echo "checked";?>
value="other">Other
```

PHP - Complete Form Example

Here is the complete code for the PHP Form Validation Example:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    } else {
        $name = test_input($_POST["name"]);
        // check if name only contains letters and whitespace
        if (!preg_match("/^[a-zA-Z ]*$/", $name)) {
            $nameErr = "Only letters and white space allowed";
        }
    }
}
```

```

if (empty($_POST["email"])) {
    $emailErr = "Email is required";
} else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $emailErr = "Invalid email format";
    }
}

if (empty($_POST["website"])) {
    $website = "";
} else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular
    // expression also allows dashes in the URL)
    if
    (!preg_match("/\b(?:(:?https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?~_!
    !:.,;]*[-a-z0-9+&@#\/%~_]/i",$website)) {
        $websiteErr = "Invalid URL";
    }
}

if (empty($_POST["comment"])) {
    $comment = "";
} else {
    $comment = test_input($_POST["comment"]);
}

if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
} else {
    $gender = test_input($_POST["gender"]);
}
}

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>

```

<h2>PHP Form Validation Example</h2>

```

<p><span class="error">* required field</span></p>
<form method="post" action="<?php
echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
    Name: <input type="text" name="name" value="<?php echo
$name;?>">
        <span class="error">* <?php echo $nameErr;?></span>
        <br><br>
    E-mail: <input type="text" name="email" value="<?php echo
$email;?>">
        <span class="error">* <?php echo $emailErr;?></span>
        <br><br>
    Website: <input type="text" name="website" value="<?php echo
$website;?>">
        <span class="error"><?php echo $websiteErr;?></span>
        <br><br>
    Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>
        <br><br>
    Gender:
        <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="female") echo "checked";?> value="female">Female
        <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="male") echo "checked";?> value="male">Male
        <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="other") echo "checked";?> value="other">Other
        <span class="error">* <?php echo $genderErr;?></span>
        <br><br>
        <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>

```

PHP7 Advanced

Multidimensional Arrays

Earlier in this tutorial, we have described arrays that are a single list of key/value pairs. However, sometimes you want to store values with more than one key. This can be stored in multidimensional arrays.

A multidimensional array is an array containing one or more arrays. PHP understands multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array  
(  
    array("Volvo",22,18),  
    array("BMW",15,13),  
    array("Saab",5,2),  
    array("Land Rover",17,15)  
);
```

Now the two-dimensional \$cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the \$cars array we must point to the two indices (row and column):

Example

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2]."<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2]."<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2]."<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2]."<br>";
?>
```

We can also put a for loop inside another for loop to get the elements of the \$cars array (we still have to point to the two indices):

Example

```
<?php
for ($row = 0; $row < 4; $row++) {
    echo "<p><b>Row number $row</b></p>";
    echo "<ul>";
    for ($col = 0; $col < 3; $col++) {
        echo "<li>".$cars[$row][$col]."</li>";
    }
    echo "</ul>";
}
?>
```

Include Files

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

PHP include and require Statements

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

The include and require statements are identical, except upon failure:

- require will produce a fatal error (E_COMPILE_ERROR) and stop the script
- include will only produce a warning (E_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of Framework, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

Syntax

include 'filename';

or

require 'filename';

PHP include Examples

Example 1

Assume we have a standard footer file called "footer.php", that looks like this:

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";
?>
```

To include the footer file in a page, use the include statement:

Example

```
<html>
```

```
<body>
```

```
<h1>Welcome to my home page!</h1>
```

```
<p>Some text.</p>
```

```
<p>Some more text.</p>
```

```
<?php include 'footer.php';?>
```

```
</body>
```

```
</html>
```

Example 2

Assume we have a standard menu file called "menu.php":

```
<?php
echo '<a href="/default.asp">Home</a> -
<a href="/html/default.asp">HTML Tutorial</a> -
<a href="/css/default.asp">CSS Tutorial</a> -
<a href="/js/default.asp">JavaScript Tutorial</a> -
<a href="default.asp">PHP Tutorial</a>';
?>
```

All pages in the Web site should use this menu file. Here is how it can be done (we are using a <div> element so that the menu easily can be styled with CSS later):

Example

```
<html>
```

```
<body>
```

```
<div class="menu">
<?php include 'menu.php';?>
</div>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>

</body>
</html>
```

Example 3

Assume we have a file called "vars.php", with some variables defined:

```
<?php
$color='red';
$car='BMW';
?>
```

Then, if we include the "vars.php" file, the variables can be used in the calling file:

Example

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php include 'vars.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

PHP include vs. require

The require statement is also used to include a file into the PHP code.

However, there is one big difference between include and require; when a file is included with the include statement and PHP cannot find it, the script will continue to execute:

Example

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php include 'noFileExists.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

If we do the same example using the require statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error:

Example

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php require 'noFileExists.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

Use **require** when the file is required by the application.

Use **include** when the file is not required and application should continue when file is not found.

File Handling

File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

Be careful when manipulating files!

When you are manipulating files you must be very careful. You can do a lot of damage if you do something wrong.

Common errors are:

- editing the wrong file,
- filling a hard-drive with garbage data,
- and deleting the content of a file by accident.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML

CSS = Cascading Style Sheets

HTML = Hyper Text Markup Language

PHP = PHP Hypertext Preprocessor

SQL = Structured Query Language

SVG = Scalable Vector Graphics

XML = EXtensible Markup Language

The PHP code to read the file and write it to the output buffer is as follows (the `readfile()` function returns the number of bytes read on success):

Example

```
<?php
echo readfile("webdictionary.txt");
?>
```

The `readfile()` function is useful if all you want to do is open up a file and read its contents.

File Open/Read/Close

In this chapter we will teach you how to open, read, and close a file on the server.

PHP Open File - `fopen()`

A better method to open files is with the `fopen()` function. This function gives you more options than the `readfile()` function.

We will use the text file, "webdictionary.txt", during the lessons:

AJAX = Asynchronous JavaScript and XML

CSS = Cascading Style Sheets

HTML = Hyper Text Markup Language

PHP = PHP Hypertext Preprocessor

SQL = Structured Query Language

SVG = Scalable Vector Graphics

XML = EXtensible Markup Language

The first parameter of `fopen()` contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the `fopen()` function is unable to open the specified file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

Tip: The `fread()` and the `fclose()` functions will be explained below.

The file may be opened in one of the following modes:

Mode	Description
------	-------------

r	Open a file for read only. File pointer starts at the beginning of the file
---	--

w	Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	Creates a new file for write only. Returns FALSE and an error if file already exists
r+	Open a file for read/write. File pointer starts at the beginning of the file
w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Close File - fclose()

The fclose() function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

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

PHP Read Single Line - fgets()

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

Note: After a call to the `fgets()` function, the file pointer has moved to the next line.

PHP Check End-Of-File - feof()

The feof() function checks if the "end-of-file" (EOF) has been reached.

The feof() function is useful for looping through data of unknown length.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
    echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
    echo fgetc($myfile);
}
fclose($myfile);
?>
```

Note: After a call to the `fgetc()` function, the file pointer moves to the next character.

File Create/Write

In this chapter we will teach you how to create and write to a file on the server.

PHP Create File - fopen()

The `fopen()` function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use `fopen()` on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

Example

```
$myfile = fopen("testfile.txt", "w")
```

PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

PHP Write to File - fwrite()

The `fwrite()` function is used to write to a file.

The first parameter of `fwrite()` contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string `$txt` that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the `fclose()` function.

If we open the "newfile.txt" file it would look like this:

John Doe
Jane Doe

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote is present:

Mickey Mouse
Minnie Mouse

File Upload

With PHP, it is easy to upload files to the server.

However, with ease comes danger, so always be careful when allowing file uploads!

Configure The "php.ini" File

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the file_uploads directive, and set it to On:

file_uploads = On

Create The HTML Form

Next, create an HTML form that allow users to choose the image file they want to upload:

```
<!DOCTYPE html>
<html>
<body>

<form action="upload.php" method="post" enctype="multipart/form-data">
  Select image to upload:
  <input type="file" name="fileToUpload" id="fileToUpload">
  <input type="submit" value="Upload Image" name="submit">
</form>
```



```
</body>
</html>
```

Some rules to follow for the HTML form above:

- Make sure that the form uses method="post"
- The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form

Without the requirements above, the file upload will not work.

Other things to notice:

- The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control
- The form above sends data to a file called "upload.php", which we will create next.

Create The Upload File PHP Script

The "upload.php" file contains the code for uploading a file:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
?>
```

PHP script explained:

- \$target_dir = "uploads/" - specifies the directory where the file is going to be placed
- \$target_file specifies the path of the file to be uploaded
- \$uploadOk=1 is not used yet (will be used later)
- \$imageFileType holds the file extension of the file (in lower case)
- Next, check if the image file is an actual image or a fake image

Note: You will need to create a new directory called "uploads" in the directory where "upload.php" file resides. The uploaded files will be saved there.

Check if File Already Exists

Now we can add some restrictions.

First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and \$uploadOk is set to 0:

```
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
```

Limit File Size

The file input field in our HTML form above is named "fileToUpload".

Now, we want to check the size of the file. If the file is larger than 500KB, an error message is displayed, and \$uploadOk is set to 0:

```
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
```

Limit File Type

The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting \$uploadOk to 0:

```
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
```

Complete Upload File PHP Script

The complete "upload.php" file now looks like this:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
    }
}
```

```

        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file ". basename( $_FILES["fileToUpload"]["name"]). " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>

```

PHP 7 Cookies

A cookie is often used to identify a user.

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the `setcookie()` function.

Syntax

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

Only the name parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable \$_COOKIE). We also use the isset() function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>

<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Note: The setcookie() function must appear BEFORE the <html> tag.

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
```

```

<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>

```

Delete a Cookie

To delete a cookie, use the `setcookie()` function with an expiration date in the past:

Example

```

<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>

<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>

```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the `setcookie()` function, then count the `$_COOKIE` array variable:

Example

```

<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>

<html>
<body>

<?php
if(count($_COOKIE) > 0) {
    echo "Cookies are enabled.";
}

```

```
} else {  
    echo "Cookies are disabled.";  
}  
?>
```

```
</body>  
</html>
```

Sessions

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.

Tip: If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session

A session is started with the `session_start()` function.
Session variables are set with the PHP global variable: `$_SESSION`.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```
<?php  
// Start the session  
session_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// Set session variables  
$_SESSION["favcolor"] = "green";  
$_SESSION["favanimal"] = "cat";  
echo "Session variables are set.";
```

```
?>
```

```
</body>
```

```
</html>
```

Note: The `session_start()` function must be the very first thing in your document. Before any HTML tags

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (`session_start()`).

Also notice that all session variable values are stored in the global `$_SESSION` variable:

Example

```
<?php
```

```
session_start();
```

```
?>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

```
// Echo session variables that were set on previous page
```

```
echo "Favorite color is " . $_SESSION["favcolor"] . "<br>";
```

```
echo "Favorite animal is " . $_SESSION["favanimal"] . " .";
```

```
?>
```

```
</body>
```

```
</html>
```

Another way to show all the session variable values for a user session is to run the following code:

Example

```
<?php
```

```
session_start();
```

```
?>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

```
print_r($_SESSION);
```

```
?>
```

```
</body>
</html>
```

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this: **765487cf34ert8dede5a562e4f3a7e12**. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>
```



```
</body>
</html>
```

Filters

Validating data = Determine if the data is in proper form.

Sanitizing data = Remove any illegal character from the data.

The PHP Filter Extension

PHP filters are used to validate and sanitize external input.

The PHP filter extension has many of the functions needed for checking user input, and is designed to make data validation easier and quicker.

The `filter_list()` function can be used to list what the PHP filter extension offers:

Example

```
<table>
<tr>
  <td>Filter Name</td>
  <td>Filter ID</td>
</tr>
<?php
foreach (filter_list() as $id => $filter) {
  echo '<tr><td>' . $filter . '</td><td>' . filter_id($filter) . '</td></tr>';
}
?>
</table>
```

Why Use Filters?

Many web applications receive external input. External input/data can be:

- User input from a form
- Cookies
- Web services data
- Server variables
- Database query results

You should always validate external data!

Invalid submitted data can lead to security problems and break your webpage!

By using PHP filters you can be sure your application gets the correct input!

PHP `filter_var()` Function

The `filter_var()` function both validate and sanitize data.

The `filter_var()` function filters a single variable with a specified filter. It takes two pieces of data:

The variable you want to check

The type of check to use

Sanitize a String

The following example uses the `filter_var()` function to remove all HTML tags from a string:
Example

```
<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;
?>
```

Validate an Integer

The following example uses the `filter_var()` function to check if the variable `$int` is an integer. If `$int` is an integer, the output of the code below will be: "Integer is valid". If `$int` is not an integer, the output will be: "Integer is not valid":

Example

```
<?php
$int = 100;

if (!filter_var($int, FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Tip: `filter_var()` and Problem With 0

In the example above, if `$int` was set to 0, the function above will return "Integer is not valid". To solve this problem, use the code below:

Example

```
<?php
$int = 0;

if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int, FILTER_VALIDATE_INT)
=== false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Validate an IP Address

The following example uses the `filter_var()` function to check if the variable `$ip` is a valid IP address:

Example

```
<?php
$ip = "127.0.0.1";
```

```
if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {  
    echo("$ip is a valid IP address");  
} else {  
    echo("$ip is not a valid IP address");  
}  
?>
```

Sanitize and Validate an Email Address

The following example uses the `filter_var()` function to first remove all illegal characters from the `$email` variable, then check if it is a valid email address:

Example

```
<?php  
$email = "john.doe@example.com";  
  
// Remove all illegal characters from email  
$email = filter_var($email, FILTER_SANITIZE_EMAIL);  
  
// Validate e-mail  
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {  
    echo("$email is a valid email address");  
} else {  
    echo("$email is not a valid email address");  
}  
?>
```

Sanitize and Validate a URL

The following example uses the `filter_var()` function to first remove all illegal characters from a URL, then check if `$url` is a valid URL:

Example

```
<?php  
$url = "https://www.w3schools.com";  
  
// Remove all illegal characters from a url  
$url = filter_var($url, FILTER_SANITIZE_URL);  
  
// Validate url  
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {  
    echo("$url is a valid URL");  
} else {  
    echo("$url is not a valid URL");  
}  
?>
```

Filters Advanced

Validate an Integer Within a Range

The following example uses the `filter_var()` function to check if a variable is both of type INT, and between 1 and 200:

Example

```
<?php
$int = 122;
$min = 1;
$max = 200;

if (filter_var($int, FILTER_VALIDATE_INT, array("options" => array("min_range"=>$min,
"max_range"=>$max))) === false) {
    echo("Variable value is not within the legal range");
} else {
    echo("Variable value is within the legal range");
}
?>
```

Validate IPv6 Address

The following example uses the `filter_var()` function to check if the variable `$ip` is a valid IPv6 address:

Example

```
<?php
$ip = "2001:0db8:85a3:08d3:1319:8a2e:0370:7334";

if (!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6) === false) {
    echo("$ip is a valid IPv6 address");
} else {
    echo("$ip is not a valid IPv6 address");
}
?>
```

Validate URL - Must Contain QueryString

The following example uses the `filter_var()` function to check if the variable `$url` is a URL with a querystring:

Example

```
<?php
$url = "https://www.w3schools.com";

if (!filter_var($url, FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED) ===
false) {
    echo("$url is a valid URL with a query string");
} else {
```

```
    echo("$url is not a valid URL with a query string");  
}  
?>
```

Remove Characters With ASCII Value > 127

The following example uses the `filter_var()` function to sanitize a string. It will both remove all HTML tags, and all characters with ASCII value > 127, from the string:

Example

```
<?php  
$str = "<h1>Hello World/ÆØÅ!</h1>";
```

```
$newstr = filter_var($str, FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_HIGH);  
echo $newstr;  
?>
```

MySQL Database

With PHP, you can connect to and manipulate databases.
MySQL is the most popular database system used with PHP.

What is MySQL?

- a database system used on the web
- a database system that runs on a server
- ideal for both small and large applications
- very fast, reliable, and easy to use
- uses standard SQL
- compiles on a number of platforms
- free to download and use
- developed, distributed, and supported by Oracle Corporation
- named after co-founder Monty Widenius's daughter: My

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

Databases are useful for storing information categorically. A company may have a database with the following tables:

- Employees
- Products
- Customers
- Orders

PHP + MySQL Database System

PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

Connect to MySQL

PHP 5 and later can work with a MySQL database using:

- MySQLi extension (the "i" stands for improved)
- PDO (PHP Data Objects)

Open a Connection to MySQL

Before we can access data in the MySQL database, we need to be able to connect to the server:

Example (MySQLi Object-Oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

PHP is an amazing and popular language!

Note on the object-oriented example above: `$connect_error` was broken until PHP 5.2.9 and 5.3.0. If you need to ensure compatibility with PHP versions prior to 5.2.9 and 5.3.0, use the following code instead:

```
// Check connection
if (mysqli_connect_error()) {
    die("Database connection failed: " . mysqli_connect_error());
}
```

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
```

```

}
echo "Connected successfully";
?>

```

Example (PDO)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
    $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
}
catch(PDOException $e)
{
    echo "Connection failed: " . $e->getMessage();
}
?>

```

Note: In the PDO example above we have also specified a database (myDB). PDO require a valid database to connect to. If no database is specified, an exception is thrown.

Tip: A great benefit of PDO is that it has an exception class to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

Close the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

Example (MySQLi Object-Oriented)

```
$conn->close();
```

Example (MySQLi Procedural)

```
mysqli_close($conn);
```

Example (PDO)

```
$conn = null;
```

Create a MySQL Database

A database consists of one or more tables.

You will need special CREATE privileges to create or to delete a MySQL database.

Create a MySQL Database Using MySQLi and PDO

The CREATE DATABASE statement is used to create a database in MySQL.

The following examples create a database named "myDB":

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn->error;
}

$conn->close();
?>
```

Note: When you create a new database, you must only specify the first three arguments to the mysqli object (servername, username and password).

Tip: If you have to use a specific port, add an empty string for the database-name argument, like this: **new mysqli("localhost", "username", "password", "", port)**

Create MySQL Tables

A database table has its own unique name and consists of columns and rows.

Create a MySQL Table Using MySQLi

The CREATE TABLE statement is used to create a table in MySQL.

We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
```



```
firstname VARCHAR(30) NOT NULL,  
lastname VARCHAR(30) NOT NULL,  
email VARCHAR(50),  
reg_date TIMESTAMP  
)
```

Notes on the table above:

The data type specifies what type of data the column can hold. For a complete reference of all the available data types, go to our Data Types reference.

After the data type, you can specify other optional attributes for each column:

1. NOT NULL - Each row must contain a value for that column, null values are not allowed
2. DEFAULT value - Set a default value that is added when no other value is passed
3. UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
4. AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
5. PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT
6. Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

The following examples shows how to create the table in PHP:

Example (MySQLi Object-oriented)

```
<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
$dbname = "myDB";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password, $dbname);  
// Check connection  
if ($conn->connect_error) {  
    die("Connection failed: " . $conn->connect_error);  
}  
  
// sql to create table  
$sql = "CREATE TABLE MyGuests (  
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
firstname VARCHAR(30) NOT NULL,  
lastname VARCHAR(30) NOT NULL,  
email VARCHAR(50),  
reg_date TIMESTAMP
```

```

);

if ($conn->query($sql) === TRUE) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . $conn->error;
}

$conn->close();
?>

```

Insert Multiple Records Into MySQL

Using MySQLi

Multiple SQL statements must be executed with the `mysqli_multi_query()` function.

The following examples add three new records to the "MyGuests" table:

Example (MySQLi Object-oriented)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com')";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if ($conn->multi_query($sql) === TRUE) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>

```

Note that each SQL statement must be separated by a semicolon.

Get ID of Last Inserted Record

If we perform an INSERT or UPDATE on a table with an AUTO_INCREMENT field, we can get the ID of the last inserted/updated record immediately.

In the table "MyGuests", the "id" column is an AUTO_INCREMENT field:

```
CREATE TABLE MyGuests (  
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
firstname VARCHAR(30) NOT NULL,  
lastname VARCHAR(30) NOT NULL,  
email VARCHAR(50),  
reg_date TIMESTAMP  
)
```

The following examples are equal to the examples from the previous page (PHP Insert Data Into MySQL), except that we have added one single line of code to retrieve the ID of the last inserted record. We also echo the last inserted ID:

Example (MySQLi Object-oriented)

```
<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
$dbname = "myDB";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password, $dbname);  
// Check connection  
if ($conn->connect_error) {  
    die("Connection failed: " . $conn->connect_error);  
}  
  
$sql = "INSERT INTO MyGuests (firstname, lastname, email)  
VALUES ('John', 'Doe', 'john@example.com')";  
  
if ($conn->query($sql) === TRUE) {  
    $last_id = $conn->insert_id;  
    echo "New record created successfully. Last inserted ID is: " . $last_id;  
} else {  
    echo "Error: " . $sql . "<br>" . $conn->error;  
}  
  
$conn->close();  
?>
```

Prepared Statements

Prepared statements are very useful against SQL injections.

Prepared Statements and Bound Parameters

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

Compared to executing SQL statements directly, prepared statements have three main advantages:

- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

Prepared Statements in MySQLi

The following example uses **prepared statements** and **bound parameters** in MySQLi:

Example (MySQLi with Prepared Statements)

```
<?php
```

```
$servername = "localhost";
```

```
$username = "username";
```

```
$password = "password";
```

```
$dbname = "myDB";
```

```
// Create connection
```

```
$conn = new mysqli($servername, $username, $password, $dbname);
```

```
// Check connection
```

```
if ($conn->connect_error) {
```

```

    die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>

```

Code lines to explain from the example above:

"INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)"

In our SQL, we insert a question mark (?) where we want to substitute in an integer, string, double or blob value.

Then, have a look at the bind_param() function:

```
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

This function binds the parameters to the SQL query and tells the database what the parameters are. The "sss" argument lists the types of data that the parameters are. The s character tells mysql that the parameter is a string.

The argument may be one of four types:

- i - integer
- d - double
- s - string

b - BLOB

We must have one of these for each parameter.

By telling mysql what type of data to expect, we minimize the risk of SQL injections.

Note: If we want to insert any data from external sources (like user input), it is very important that the data is sanitized and validated.

Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

SELECT column_name(s) FROM table_name
or we can use the * character to select ALL columns from a table:

SELECT * FROM table_name

Select Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:

```
<!DOCTYPE html>
<html>
<body>

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "<br> id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"] . "<br>";
    }
} else {
    echo "0 results";
}
```

```
$conn->close();  
?>
```

```
</body>  
</html>
```

Code lines to explain from the example above:

First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The next line of code runs the query and puts the resulting data into a variable called \$result.

Then, the function num_rows() checks if there are more than zero rows returned.

If there are more than zero rows returned, the function fetch_assoc() puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.

You can also put the result in an HTML table:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
table, th, td {
```

```
    border: 1px solid black;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<?php
```

```
$servername = "localhost";
```

```
$username = "username";
```

```
$password = "password";
```

```
$dbname = "myDB";
```

```
// Create connection
```

```
$conn = new mysqli($servername, $username, $password, $dbname);
```

```
// Check connection
```

```
if ($conn->connect_error) {
```

```
    die("Connection failed: " . $conn->connect_error);
```

```
}
```

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
```

```
$result = $conn->query($sql);
```

```

if ($result->num_rows > 0) {
    echo "<table><tr><th>ID</th><th>Name</th></tr>";
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "<tr><td>" . $row["id"]. "</td><td>" . $row["firstname"]. " " . $row["lastname"].
"</td></tr>";
    }
    echo "</table>";
} else {
    echo "0 results";
}

$conn->close();
?>

</body>
</html>

```

Delete Data From a MySQL Table Using MySQLi

The **DELETE** statement is used to delete records from a table:

```
DELETE FROM table_name
```

```
WHERE some_column = some_value
```

Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

The following examples delete the record with id=3 in the "MyGuests" table:

Example (MySQLi Object-oriented)

```
<?php
```

```
$servername = "localhost";
```

```
$username = "username";
```

```
$password = "password";
```



```

$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . $conn->error;
}

$conn->close();
?>

```

Update Data In a MySQL Table Using MySQLi

The **UPDATE** statement is used to update existing records in a table:

```

UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value

```

Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

The following examples update the record with id=2 in the "MyGuests" table:

Example (MySQLi Object-oriented)

```
<?php
```

```

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if ($conn->query($sql) === TRUE) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . $conn->error;
}

$conn->close();
?>

```

Limit Data Selections From a MySQL Database

MySQL provides a LIMIT clause that is used to specify the number of records to return.

The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.

Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

```
$sql = "SELECT * FROM Orders LIMIT 30";
```

When the SQL query above is run, it will return the first 30 records.

What if we want to select records 16 - 25 (inclusive)?

Mysql also provides a way to handle this: by using OFFSET.

The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":

```
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
```

You could also use a shorter syntax to achieve the same result:

```
$sql = "SELECT * FROM Orders LIMIT 15, 10";
```

Notice that the numbers are reversed when you use a comma.

