PHP 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.

PHP 7 is the latest stable release.

Easy Learning with "PHP Tryit"

With our online "PHP Tryit" editor, you can edit the PHP code, and click on a button to view the result.

Example

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

<?php
echo "My first PHP script!";
?>

</body>
</html>
```

Click on the "Try it Yourself" button to see how it works.

PHP Exercises

PHP Examples

Learn by examples! This tutorial supplements all explanations with clarifying examples.

PHP References

W3Schools' PHP reference contains different categories of all PHP functions and constants, along with examples.

- Array
- Calendar
- Date
- Directory
- Error
- Filesystem
- Filter
- FTP
- Libxml
- Mail

PHP Introduction

PHP code is executed on the server.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- CSS
- JavaScript

If you want to study these subjects first, find the tutorials on our Home page.

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 is 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

What's new in PHP 7

- PHP 7 is much faster than the previous popular stable release (PHP 5.6)
- PHP 7 has improved Error Handling

- PHP 7 supports stricter Type Declarations for function arguments
- PHP 7 supports new operators (like the spaceship operator: <=>)

PHP 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:

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 (;).

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 equal and legal:

Example

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

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

</body>
</html>
```

Note: However; all variable names are case-sensitive!

Look at 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>
```

PHP Exercises

Test Yourself With Exercises

PHP Comments

Comments in PHP

A comment in PHP code is a line that is not executed as a 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 your code
- 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

Syntax for single-line comments:

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

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

# This is also a single-line comment
?>

</body>
</html>
```

Example

Syntax for multiple-line comments:

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

<?php
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
?>

</body>
</html>
```

Example

Using comments to leave out parts of the code:

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

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

</body>
</html>
```

PHP 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
<?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 that 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:

Example

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

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

Example

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

The following example will output the sum of two variables:

Example

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

Note: You will learn more about the **echo** statement and how to output data to the screen in the next chapter.

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 associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives 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.

You will learn more about **strict** and **non-strict** requirements, and data type declarations in the PHP Functions chapter.

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

Variable with global scope:

```php
<?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

Variable with local scope:

```php
<?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
<?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
<?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

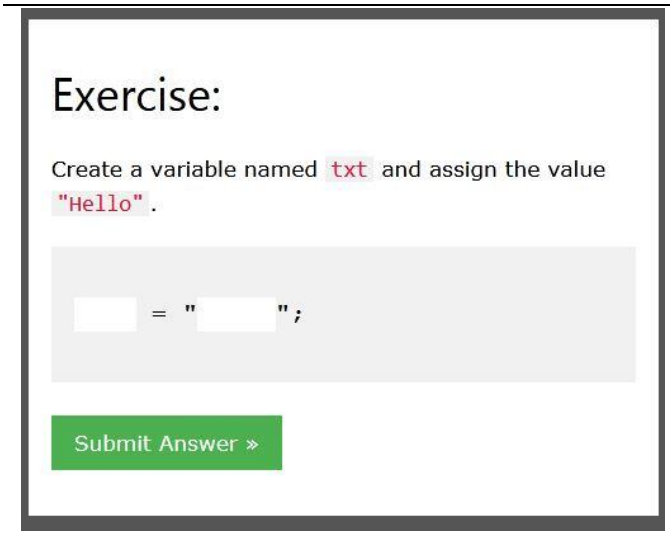
```php
<?php
function myTest() {
    static $x = 0;
    echo $x;
    $x++;
}

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.

PHP Exercises

Test Yourself With Exercises



PHP echo and print Statements

With PHP, there are two basic ways to get output: **echo** and **print**.

In this tutorial we use **echo** or **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):

Example

```
<?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:

Example

```
<?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:

Example

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

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

PHP Data Types

PHP 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
- Resource

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
<?php
$x = "Hello world!";
$y = 'Hello world!';

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

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: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

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

Example

```php
<?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:

Example

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

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```php
$x = true;
$y = false;
```

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

PHP 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:

Example

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

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

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
<?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
<?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.

We will not talk about the resource type here, since it is an advanced topic.

PHP Strings

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

PHP String Functions

In this chapter we will look at some commonly used functions to manipulate strings.

strlen() - Return the Length of a String

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

Example

Return the length of the string "Hello world!":

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

str_word_count() - Count Words in a String

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

Example

Count the number of word in the string "Hello world!":

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

strrev() - Reverse a String

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

Example

Reverse the string "Hello world!":

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

strpos() - Search For a 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.

Example

Search for the text "world" in the string "Hello world!":

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

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

str_replace() - Replace Text Within a String

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

Example

Replace the text "world" with "Dolly":

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

Complete PHP String Reference

For a complete reference of all string functions, go to our complete PHP String Reference.

The PHP string reference contains description and example of use, for each function!

PHP Exercises

Test Yourself With Exercises

PHP Numbers

In this chapter we will look in depth into Integers, Floats, and Number Strings.

PHP Numbers

One thing to notice about PHP is that it provides automatic data type conversion.

So, if you assign an integer value to a variable, the type of that variable will automatically be an integer. Then, if you assign a string to the same variable, the type will change to a string.

This automatic conversion can sometimes break your code.

PHP Integers

An integer is a number without any decimal part.

2, 256, -256, 10358, -179567 are all integers. While 7.56, 10.0, 150.67 are floats.

So, an integer data type is a non-decimal number between -2147483648 and 2147483647. A value greater (or lower) than

this, will be stored as float, because it exceeds the limit of an integer.

Another important thing to know is that even if 4 * 2.5 is 10, the result is stored as float, because one of the operands is a float (2.5).

Here are some 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)

PHP has the following functions to check if the type of a variable is integer:

- is_int()
- is_integer() - alias of is_int()
- is_long() - alias of is_int()

Example

Check if the type of a variable is integer:

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

$x = 59.85;
var_dump(is_int($x));
?>
```

PHP Floats

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

2.0, 256.4, 10.358, 7.64E+5, 5.56E-5 are all floats.

The float data type can commonly store a value up to 1.7976931348623E+308 (platform dependent), and have a maximum precision of 14 digits.

PHP has the following functions to check if the type of a variable is float:

- is_float()
- is_double() - alias of is_float()

Example

Check if the type of a variable is float:

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

PHP Infinity

A numeric value that is larger than PHP_FLOAT_MAX is considered infinite.

PHP has the following functions to check if a numeric value is finite or infinite:

- is_finite()
- is_infinite()

However, the PHP var_dump() function returns the data type and value:

Example

Check if a numeric value is finite or infinite:

```
<?php
$x = 1.9e411;
var_dump($x);
?>
```

PHP NaN

NaN stands for Not a Number.

NaN is used for impossible mathematical operations.

PHP has the following functions to check if a value is not a number:

- is_nan()

However, the PHP var_dump() function returns the data type and value:

Example

Invalid calculation will return a NaN value:

```php
<?php
$x = acos(8);
var_dump($x);
?>
```

PHP Numerical Strings

The PHP is_numeric() function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise.

Example

Check if the variable is numeric:

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

$x = "5985";
var_dump(is_numeric($x));

$x = "59.85" + 100;
var_dump(is_numeric($x));

$x = "Hello";
var_dump(is_numeric($x));
?>
```

Note: From PHP 7.0: The is_numeric() function will return FALSE for numeric strings in hexadecimal form (e.g. 0xf4c3b00c), as they are no longer considered as numeric strings.

PHP Casting Strings and Floats to Integers

Sometimes you need to cast a numerical value into another data type.

The (int), (integer), or intval() function are often used to convert a value to an integer.

Example

Cast float and string to integer:

```php
<?php
// Cast float to int
$x = 23465.768;
$int_cast = (int)$x;
echo $int_cast;

echo "<br>";

// Cast string to int
$x = "23465.768";
$int_cast = (int)$x;
echo $int_cast;
?>
```

PHP Constants

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

PHP Constants

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.

A valid constant name starts with a letter or underscore (no $ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

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

Example

Create a constant with a **case-sensitive** name:

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

Example

Create a constant with a **case-insensitive** name:

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

PHP Constant Arrays

In PHP7, you can create an Array constant using the **define()** function.

Example

Create 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.

Example

This example uses a constant inside a function, even if it is defined outside the function:

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

function myTest() {
    echo GREETING;
}

myTest();
?>
```

PHP Operators

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

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.

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

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

PHP Comparison Operators

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

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y.

			Introduced in PHP 7.

PHP Increment / Decrement Operators

The PHP increment operators are used to increment a variable's value.

The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

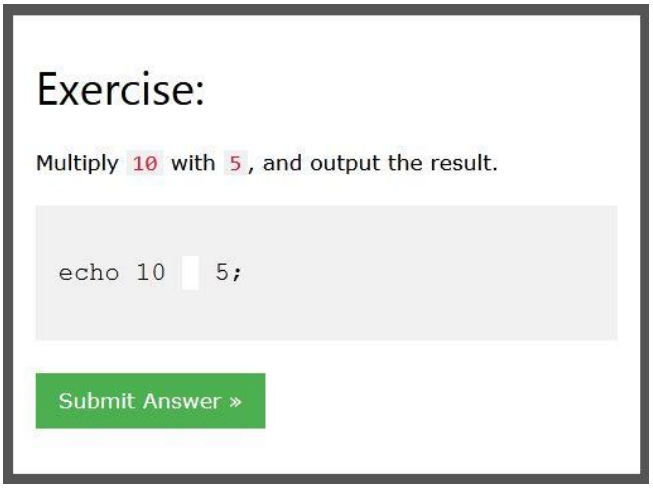
PHP Conditional Assignment Operators

The PHP conditional assignment operators are used to set a value depending on conditions:

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

PHP Exercises

Test Yourself With Exercises



PHP 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;
}
```

Example

Output "Have a good day!" if the current time (HOUR) is less than 20:

```
<?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;
}
```

Example

Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```
<?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;
}
```

Example

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!":

```php
<?php
$t = date("H");

if ($t < "10") {
    echo "Have a good morning!";
} elseif ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

PHP - The switch Statement

The **switch** statement will be explained in the next chapter.

PHP Exercises

Test Yourself With Exercises

PHP switch Statement

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

The PHP switch Statement

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

Syntax

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
    ...
    default:
        code to be executed if n is different
from all labels;
}
```

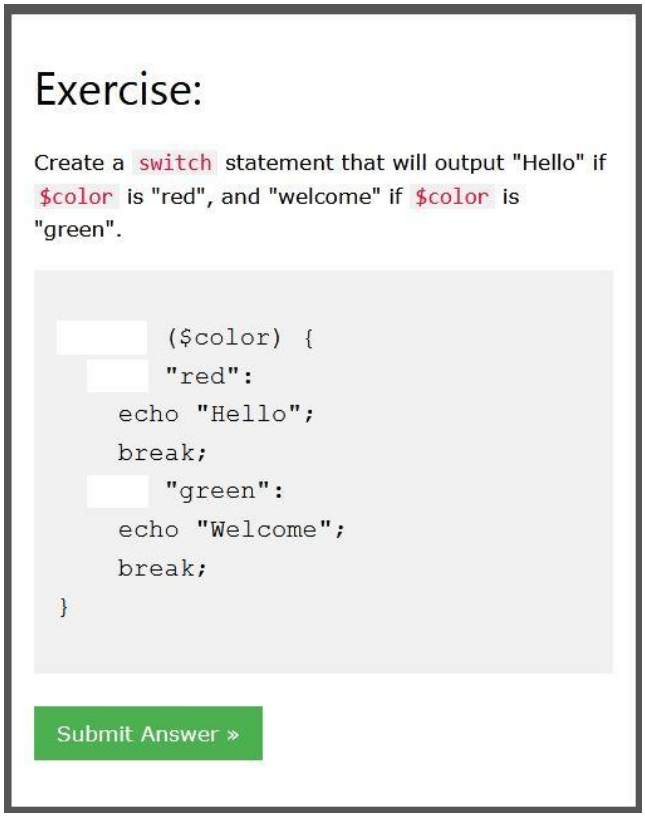
This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent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

Example

```php
<?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!";
}
?>
```

PHP Exercises

Test Yourself With Exercises



PHP Loops

In the following chapters you will learn how to repeat code by using loops in PHP.

PHP Loops

Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

Loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

- **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 following chapters will explain and give examples of each loop type.

PHP while Loop

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

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;
}
```

Examples

The example below displays the numbers from 1 to 5:

Example

```
<?php
$x = 1;

while($x <= 5) {
    echo "The number is: $x <br>";
    $x++;
}
?>
```

Example Explained

- $x = 1; - Initialize the loop counter ($x), and set the start value to 1
- $x <= 5 - Continue the loop as long as $x is less than or equal to 5
- $x++; - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens:

Example

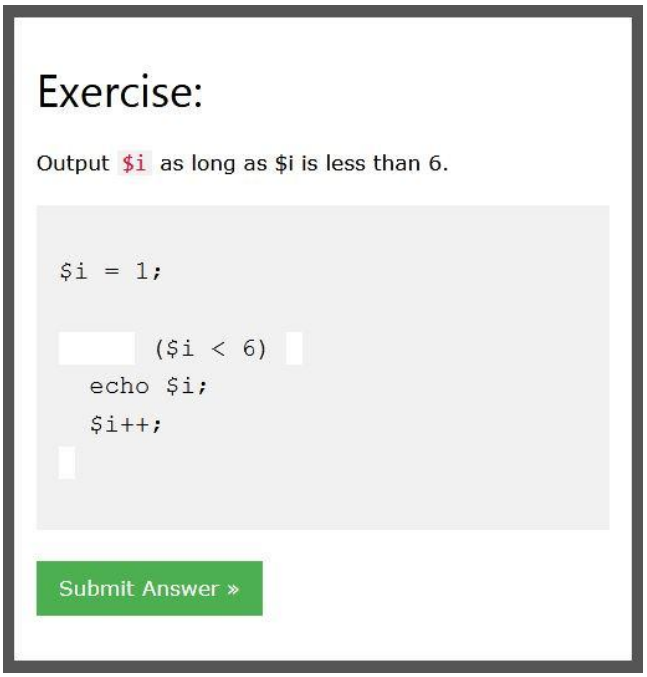
```php
<?php
$x = 0;

while($x <= 100) {
    echo "The number is: $x <br>";
    $x+=10;
}
?>
```

Example Explained

- $x = 0; - Initialize the loop counter ($x), and set the start value to 0
- $x <= 100 - Continue the loop as long as $x is less than or equal to 100
- $x+=10; - Increase the loop counter value by 10 for each iteration

PHP Exercises

Test Yourself With Exercises



PHP do while Loop

The **do...while** loop - Loops through a block of code once, and then repeats the loop as long as the specified condition is true.

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);
```

Examples

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);
?>
```

Note: In a **do...while** loop the condition is tested AFTER executing the statements within the loop. This means that the **do...while** loop will execute its statements at least once, even if the condition is false. See example below.

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

Example

```
<?php
$x = 6;

do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?>
```

PHP for Loop

The **for** loop - Loops through a block of code a specified number of times.

The PHP for Loop

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

Syntax

```
for (init counter; test counter; increment
counter) {
    code to be executed for each iteration;
}
```

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

Examples

The example below displays the numbers from 0 to 10:

Example

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

Example Explained

- $x = 0; - Initialize the loop counter ($x), and set the start value to 0
- $x <= 10; - Continue the loop as long as $x is less than or equal to 10
- $x++ - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens:

Example

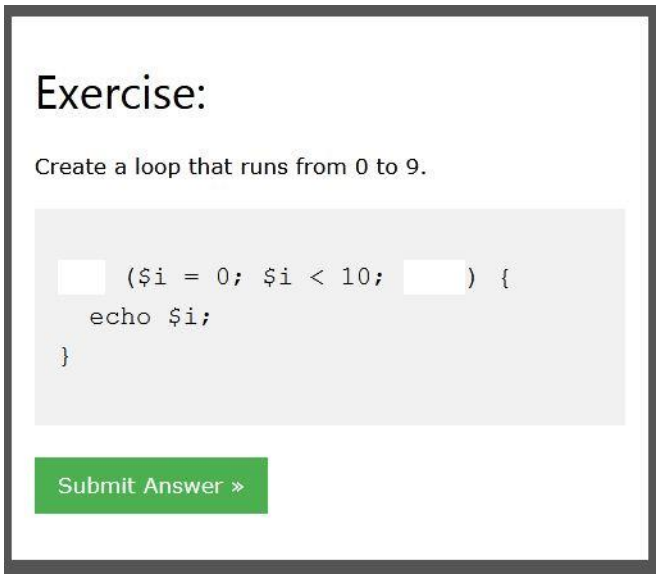
```
<?php
for ($x = 0; $x <= 100; $x+=10) {
    echo "The number is: $x <br>";
}
?>
```

Example Explained

- $x = 0; - Initialize the loop counter ($x), and set the start value to 0
- $x <= 100; - Continue the loop as long as $x is less than or equal to 100
- $x+=10 - Increase the loop counter value by 10 for each iteration

PHP Exercises

Test Yourself With Exercises



PHP foreach Loop

The **foreach** loop - Loops through a block of code for each element 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.

Examples

The following example will output the values of the given array ($colors):

Example

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

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

The following example will output both the keys and the values of the given array ($age):

Example

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

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

You will learn more about arrays in the PHP Arrays chapter.

PHP Functions

The real power of PHP comes from its functions.

PHP has more than 1000 built-in functions, and in addition you can create your own custom functions.

PHP Built-in Functions

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

Please check out our PHP reference for a complete overview of the PHP built-in functions.

PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically 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 must start with a letter or an underscore. Function names are NOT case-sensitive.

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

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

Example

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

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

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

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

Example

```
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

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

Example

```php
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

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 associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

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

In the following example we try to send both a number and a string to the function without using **strict**:

Example

```php
<?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
?>
```

To specify **strict** we need to set **declare(strict_types=1);**. This must be on the very first line of the PHP file.

In the following example we try to send both a number and a string to the function, but here we have added the **strict** declaration:

Example

```php
<?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
?>
```

The **strict** declaration 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:

Example

```php
<?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
?>
```

PHP Functions - Returning values

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

Example

```php
<?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);
?>
```

PHP Exercises

Test Yourself With Exercises

PHP Arrays

An array stores multiple values in one single variable:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . "
and " . $cars[2] . ".";
?>
```

What is an Array?

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

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

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

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

The solution is to create an array!

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

Create an Array in PHP

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

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with 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);
?>
```

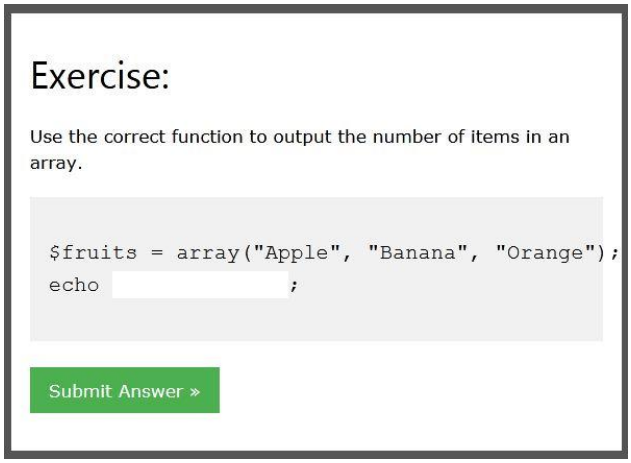
Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Exercises

Test Yourself With Exercises



PHP Indexed Arrays

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] . ".";
?>
```

Loop Through an Indexed Array

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

Example

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

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

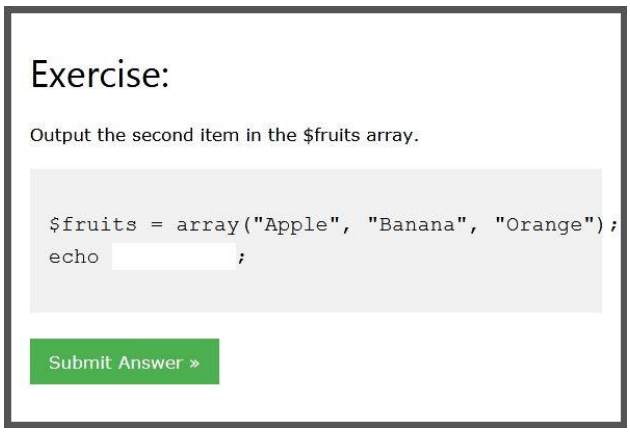
Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Exercises

Test Yourself With Exercises



PHP Associative Arrays

PHP Associative Arrays

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

There are two ways to create an associative array:

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

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example

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

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

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

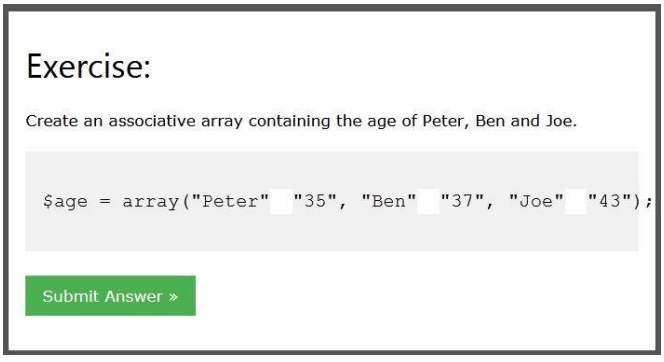
Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Exercises

Test Yourself With Exercises



PHP Multidimensional Arrays

In the previous pages, 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. For this, we have multidimensional arrays.

PHP - Multidimensional Arrays

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

PHP supports 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:

```php
$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>";
}
?>
```

Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP 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:

Example

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

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

Example

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

Sort Array in Descending Order - rsort()

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

Example

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

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

Example

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

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

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

Example

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

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

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

Example

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

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

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

Example

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

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

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

Example

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

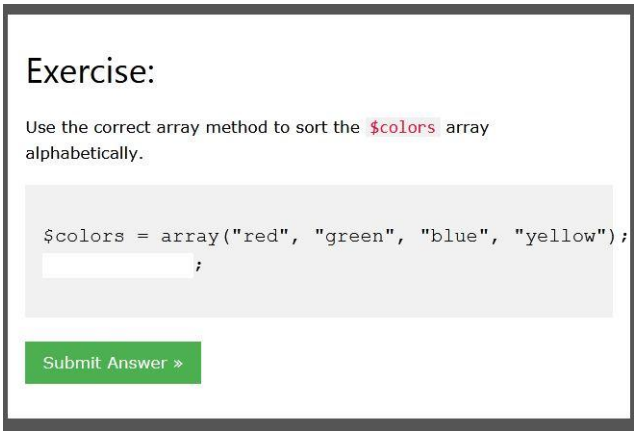
Complete PHP Array Reference

For a complete reference of all array functions, go to our complete PHP Array Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Exercises

Test Yourself With Exercises



PHP 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

Some 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

The next chapters will explain some of the superglobals, and the rest will be explained in later chapters.

PHP Superglobal - $GLOBALS

Super global variables are built-in variables that are always available in all scopes.

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:

Example

```php
<?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 Superglobal - $_SERVER

Super global variables are built-in variables that are always available in all scopes.

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
<?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
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

PHP Superglobal - $_REQUEST

Super global variables are built-in variables that are always available in all scopes.

PHP $_REQUEST

PHP $_REQUEST is a PHP super global variable which 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 Superglobal - $_POST

Super global variables are built-in variables that are always available in all scopes.

PHP $_POST

PHP $_POST is a PHP super global variable which is 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>
```

Tip: You will learn more about $_POST in the PHP Forms chapter.

PHP Superglobal - $_GET

Super global variables are built-in variables that are always available in all scopes.

PHP $_GET

PHP $_GET is a PHP super global variable which is 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":

Example

```
<html>
<body>

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

</body>
</html>
```

Tip: You will learn more about $_GET in the PHP Forms chapter.

PHP Form Handling

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

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. This can be useful in some cases.

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.

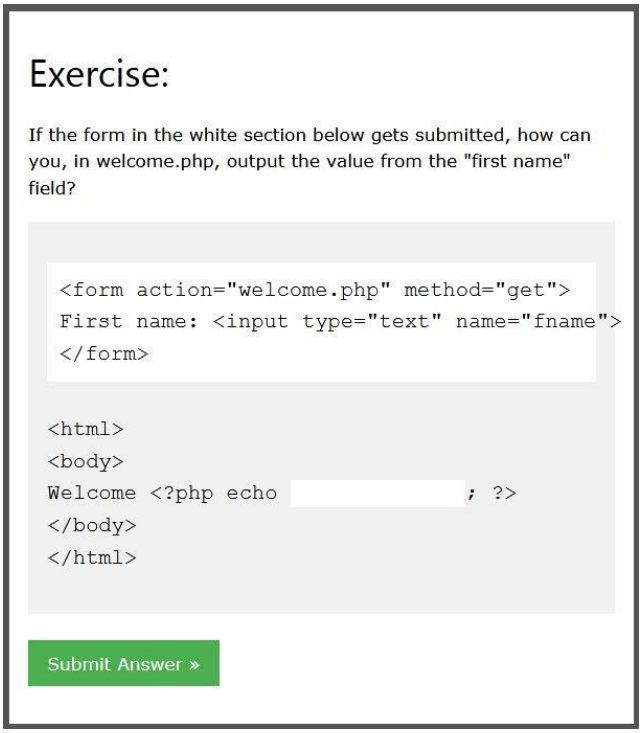
However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Developers prefer POST for sending form data.

Next, lets see how we can process PHP forms the secure way!

PHP Exercises

Test Yourself With Exercises



PHP Form Validation

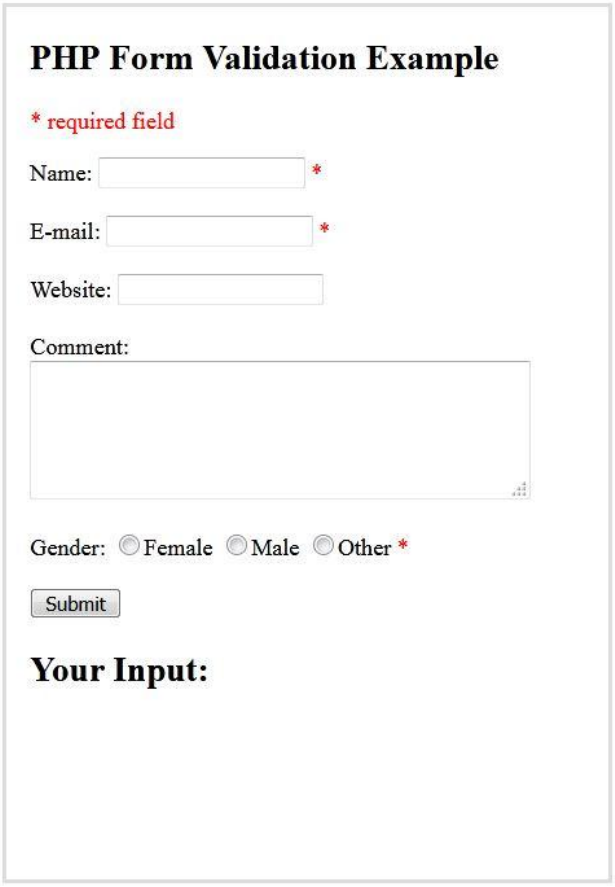
This and the next chapters show how to use PHP to validate form data.

PHP 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 HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:



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

First we will look at the plain HTML code for the form:

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:

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

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
<?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.

The next step is to make input fields required and create error messages if needed.

PHP 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
<?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

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

Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?>
</span>
<br><br>
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>
```

The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out, "Does the Website field contain a valid URL?".

PHP 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"]);
  }
}
?>
```

The next step is to show how to prevent the form from emptying all the input fields when the user submits the form.

PHP 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.

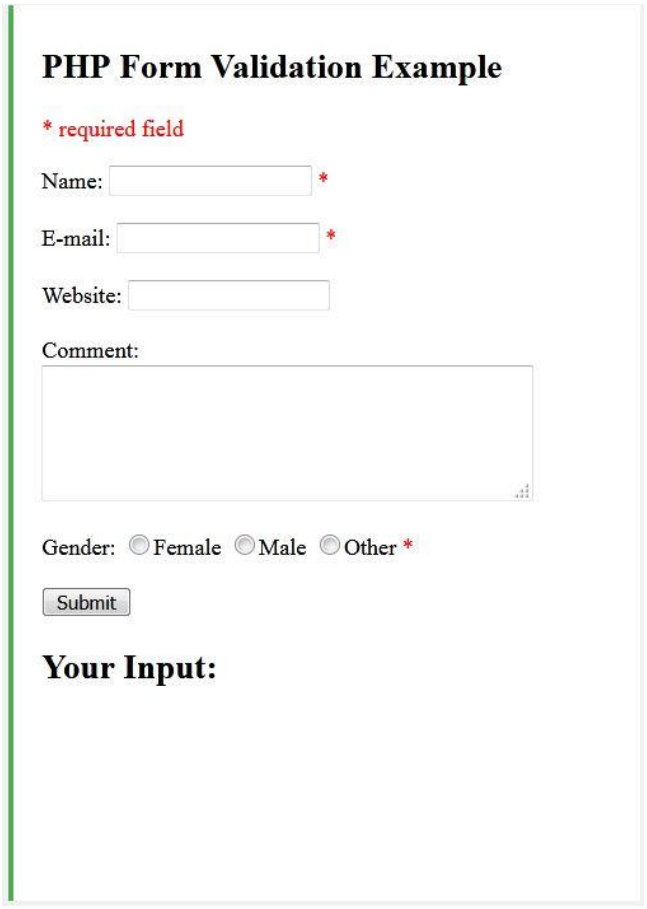
Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

```
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:

Example


PHP Form Validation Example
* required field
Name: *
E-mail: *
Website:
Comment:
Gender: Female Male Other *
Submit
Your Input:

PHP Date and Time

The PHP **date()** function is used to format a date and/or a time.

The PHP Date() Function

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

Syntax

```
date(format,timestamp)
```

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

Get a Date

The required *format* parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)
- l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

The example below formats today's date in three different ways:

Example

```
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
```

PHP Tip - Automatic Copyright Year

Use the **date()** function to automatically update the copyright year on your website:

Example

```
&copy; 2010-<?php echo date("Y");?>
```

Get a Time

Here are some characters that are commonly used for times:

- H - 24-hour format of an hour (00 to 23)
- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

Example

```
<?php
echo "The time is " . date("h:i:sa");
?>
```

Note that the PHP date() function will return the current date/time of the server!

Get Your Time Zone

If the time you got back from the code is not correct, it's probably because your server is in another country or set up for a different timezone.

So, if you need the time to be correct according to a specific location, you can set the timezone you want to use.

The example below sets the timezone to "America/New_York", then outputs the current time in the specified format:

Example

```
<?php
date_default_timezone_set("America/New_York");
echo "The time is " . date("h:i:sa");
?>
```

Create a Date With mktime()

The optional *timestamp* parameter in the date() function specifies a timestamp. If omitted, the current date and time will be used (as in the examples above).

The PHP **mktime()** function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

Syntax

```
mktime(hour, minute, second, month, day, year)
```

The example below creates a date and time with the **date()** function from a number of parameters in the **mktime()** function:

Example

```
<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is " . date("Y-m-d h:i:sa",
$d);
?>
```

Create a Date From a String With strtotime()

The PHP **strtotime()** function is used to convert a human readable date string into a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).

Syntax

```
strtotime(time, now)
```

The example below creates a date and time from the **strtotime()** function:

Example

```
<?php
$d=strtotime("10:30pm April 15 2014");
echo "Created date is " . date("Y-m-d h:i:sa",
$d);
?>
```

PHP is quite clever about converting a string to a date, so you can put in various values:

Example

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

However, **strtotime()** is not perfect, so remember to check the strings you put in there.

More Date Examples

The example below outputs the dates for the next six Saturdays:

Example

```
<?php
$startdate = strtotime("Saturday");
$enddate = strtotime("+6 weeks", $startdate);

while ($startdate < $enddate) {
  echo date("M d", $startdate) . "<br>";
  $startdate = strtotime("+1 week", $startdate);
}
?>
```

The example below outputs the number of days until 4th of July:

Example

```
<?php
$d1=strtotime("July 04");
$d2=ceil(($d1-time())/60/60/24);
echo "There are " . $d2 ." days until 4th of
July.";
?>
```

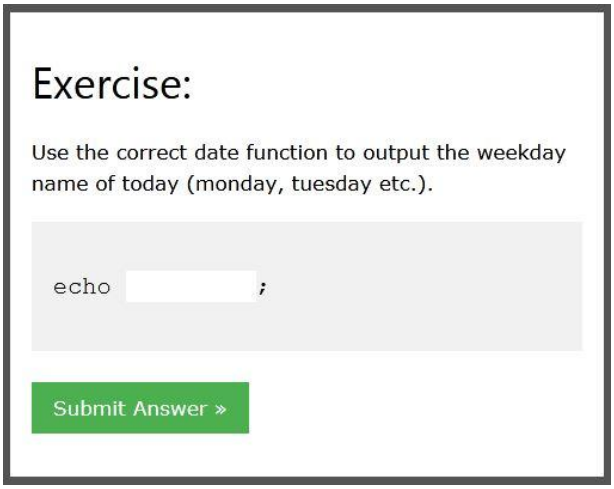
Complete PHP Date Reference

For a complete reference of all date functions, go to our complete PHP Date Reference.

The reference contains a brief description, and examples of use, for each function!

PHP Exercises

Test Yourself With Exercises



PHP 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 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

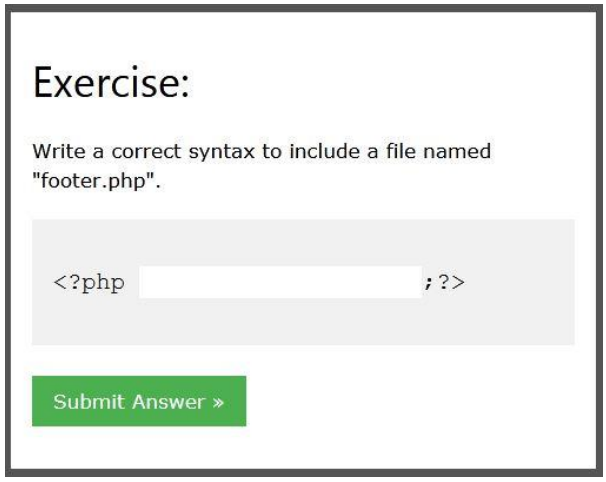
```
<html>
<body>

<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.

PHP Exercises

Test Yourself With Exercises



PHP 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.

The next chapters will teach you more about file handling.

PHP Exercises

Test Yourself With Exercises

PHP 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
<?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:

Modes	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);
?>
```

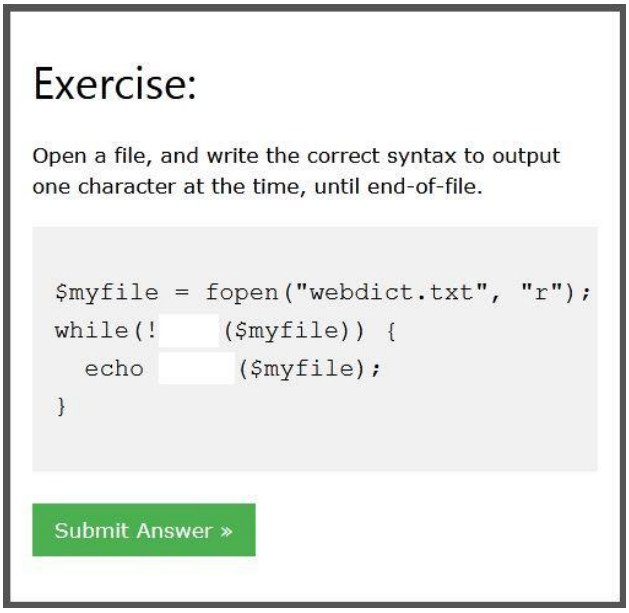
Note: After a call to the **fgetc()** function, the file pointer moves to the next character.

Complete PHP Filesystem Reference

For a complete reference of filesystem functions, go to our complete PHP Filesystem Reference.

PHP Exercises

Test Yourself With Exercises



PHP 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
<?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
```

Complete PHP Filesystem Reference

For a complete reference of filesystem functions, go to our complete PHP Filesystem Reference.

PHP 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_EXTENSI
ON));
// 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_EXTENSI
ON));
// 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.";
    }
}
?>
```

Complete PHP Filesystem Reference

For a complete reference of filesystem functions, go to our complete PHP Filesystem Reference.

PHP Cookies

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
<?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
<?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
<?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>
```

Complete PHP Network Reference

For a complete reference of Network functions, go to our complete PHP Network Reference.

PHP Exercises

Test Yourself With Exercises

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

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

// destroy the session
session_destroy();
?>

</body>
</html>
```

PHP Exercises

Test Yourself With Exercises

PHP 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");
}
?>
```

Complete PHP Filter Reference

For a complete reference of all filter functions, go to our complete PHP Filter Reference. Check each filter to see what options and flags are available.

The reference contains a brief description, and examples of use, for each function!

PHP 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 WorldÆØÅ!</h1>";

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

Complete PHP Filter Reference

For a complete reference of all filter functions, go to our complete PHP Filter Reference. Check each filter to see what options and flags are available.

The reference contains a brief description, and examples of use, for each function!

PHP and JSON

What is JSON?

JSON stands for JavaScript Object Notation, and is a syntax for storing and exchanging data.

Since the JSON format is a text-based format, it can easily be sent to and from a server, and used as a data format by any programming language.

PHP and JSON

PHP has some built-in functions to handle JSON.

First, we will look at the following two functions:

- **json_encode()**
- **json_decode()**

PHP - json_encode()

The **json_encode()** function is used to encode a value to JSON format.

Example

This example shows how to encode an associative array into a JSON object:

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

echo json_encode($age);
?>
```

Example

This example shows how to encode an indexed array into a JSON array:

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

echo json_encode($cars);
?>
```

PHP - json_decode()

The **json_decode()** function is used to decode a JSON object into a PHP object or an associative array.

Example

This example decodes JSON data into a PHP object:

```php
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

var_dump(json_decode($jsonobj));
?>
```

The **json_decode()** function returns an object by default. The **json_decode()** function has a second parameter, and when set to true, JSON objects are decoded into associative arrays.

Example

This example decodes JSON data into a PHP associative array:

```php
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

var_dump(json_decode($jsonobj, true));
?>
```

PHP - Accessing the Decoded Values

Here are two examples of how to access the decoded values from an object and from an associative array:

Example

This example shows how to access the values from a PHP object:

```php
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$obj = json_decode($jsonobj);

echo $obj->Peter;
echo $obj->Ben;
echo $obj->Joe;
?>
```

Example

This example shows how to access the values from a PHP associative array:

```php
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$arr = json_decode($jsonobj, true);

echo $arr["Peter"];
echo $arr["Ben"];
echo $arr["Joe"];
?>
```

PHP - Looping Through the Values

You can also loop through the values with a **foreach()** loop:

Example

This example shows how to loop through the values of a PHP object:

```php
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$obj = json_decode($jsonobj);

foreach($obj as $key => $value) {
  echo $key . " => " . $value . "<br>";
}
?>
```

Example

This example shows how to loop through the values of a PHP associative array:

```php
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$arr = json_decode($jsonobj, true);

foreach($arr as $key => $value) {
  echo $key . " => " . $value . "<br>";
}
?>
```

PHP - What is OOP?

From PHP5, you can also write PHP code in an object-oriented style.

Object-Oriented programming is faster and easier to execute.

PHP What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Tip: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

PHP - What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:

Another example:

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Look at the next chapters to learn more about OOP.

PHP OOP - Classes and Objects

A class is a template for objects, and an object is an instance of class.

OOP Case

Let's assume we have a class named Fruit. A Fruit can have properties like name, color, weight, etc. We can define variables like $name, $color, and $weight to hold the values of these properties.

When the individual objects (apple, banana, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Define a Class

A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods goes inside the braces:

Syntax

```php
<?php
class Fruit {
  // code goes here...
}
?>
```

Below we declare a class named Fruit consisting of two properties ($name and $color) and two methods set_name() and get_name() for setting and getting the $name property:

Example

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Note: In a class, variables are called properties and functions are called methods!

Define Objects

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Objects of a class is created using the new keyword.

In the example below, $apple and $banana are instances of the class Fruit:

Example

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

In the example below, we add two more methods to class Fruit, for setting and getting the $color property:

Example

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

PHP - The $this Keyword

The $this keyword refers to the current object, and is only available inside methods.

Look at the following example:

Example

```php
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
?>
```

So, where can we change the value of the $name property? There are two ways:

1. Inside the class (by adding a set_name() method and use $this):

Example

```php
<?php
class Fruit {
  public $name;
  function set_name($name) {
    $this->name = $name;
  }
}
$apple = new Fruit();
$apple->set_name("Apple");
?>
```

2. Outside the class (by directly change the property value):

Example

```php
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
$apple->name = "Apple";
?>
```

PHP - instanceof

You can use the **instanceof** keyword to check if an object belongs to a specific class:

Example

```php
<?php
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

PHP OOP - Constructor

PHP - The __construct Function

A constructor allows you to initialize an object's properties upon creation of the object.

If you create a **__construct()** function, PHP will automatically call this function when you create an object from a class.

Notice that the construct function starts with two underscores (__)!

We see in the example below, that using a constructor saves us from calling the set_name() method which reduces the amount of code:

Example

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

Another example:

Example

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function get_name() {
    return $this->name;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

PHP OOP - Destructor

PHP - The __destruct Function

A destructor is called when the object is destructed or the script is stopped or exited.

If you create a **__destruct()** function, PHP will automatically call this function at the end of the script.

Notice that the destruct function starts with two underscores (__)!

The example below has a __construct() function that is automatically called when you create an object from a class, and a __destruct() function that is automatically called at the end of the script:

Example

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function __destruct() {
    echo "The fruit is {$this->name}.";
  }
}

$apple = new Fruit("Apple");
?>
```

Another example:

Example

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function __destruct() {
    echo "The fruit is {$this->name} and the
color is {$this->color}.";
  }
}

$apple = new Fruit("Apple", "red");
?>
```

Tip: As constructors and destructors helps reducing the amount of code, they are very useful!

PHP OOP - Access Modifiers

PHP - Access Modifiers

Properties and methods can have access modifiers which control where they can be accessed.

There are three access modifiers:

- **public** - the property or method can be accessed from everywhere. This is default
- **protected** - the property or method can be accessed within the class and by classes derived from that class
- **private** - the property or method can ONLY be accessed within the class

In the following example we have added three different access modifiers to the three properties. Here, if you try to set the name property it will work fine (because the name property is public). However, if you try to set the color or weight property it will result in a fatal error (because the color and weight property are protected and private):

Example

```php
<?php
class Fruit {
  public $name;
  protected $color;
  private $weight;
}

$mango = new Fruit();
$mango->name = 'Mango'; // OK
$mango->color = 'Yellow'; // ERROR
$mango->weight = '300'; // ERROR
?>
```

In the next example we have added access modifiers to two methods. Here, if you try to call the set_color() or the set_weight() function it will result in a fatal error (because the two functions are considered protected and private), even if all the properties are public:

Example

```php
<?php
class Fruit {
  public $name;
  public $color;
  public $weight;

  function set_name($n) {  // a public function (default)
    $this->name = $n;
  }
  protected function set_color($n) { // a protected function
    $this->color = $n;
  }
  private function set_weight($n) { // a private function
    $this->weight = $n;
  }
}

$mango = new Fruit();
$mango->set_name('Mango'); // OK
$mango->set_color('Yellow'); // ERROR
$mango->set_weight('300'); // ERROR
?>
```

PHP OOP - Inheritance

PHP - What is Inheritance?

Inheritance in OOP = When a class derives from another class.

The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

An inherited class is defined by using the extends keyword.

Let's look at an example:

Example

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

Example Explained

The Strawberry class is inherited from the Fruit class.

This means that the Strawberry class can use the public $name and $color properties as well as the public __construct() and intro() methods from the Fruit class because of inheritance.

The Strawberry class also has its own method: message().

PHP - Inheritance and the Protected Access Modifier

In the previous chapter we learned that protected properties or methods can be accessed within the class and by classes derived from that class. What does that mean?

Let's look at an example:

Example

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the
color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}

// Try to call all three methods from outside
class
$strawberry = new Strawberry("Strawberry",
"red");  // OK. __construct() is public
$strawberry->message(); // OK. message() is
public
$strawberry->intro(); // ERROR. intro() is
protected
?>
```

In the example above we see that if we try to call a **protected** method (intro()) from outside the class, we will receive an error. **public** methods will work fine!

Let's look at another example:

Example

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the
color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
    // Call protected method from within derived
class - OK
    $this -> intro();
  }
}

$strawberry = new Strawberry("Strawberry",
"red"); // OK. __construct() is public
$strawberry->message(); // OK. message() is
public and it calls intro() (which is protected)
from within the derived class
?>
```

In the example above we see that all works fine! It is because we call the protected method (intro()) from inside the derived class.

PHP - Overriding Inherited Methods

Inherited methods can be overridden by redefining the methods (use the same name) in the child class.

Look at the example below. The __construct() and intro() methods in the child class (Strawberry) will override the __construct() and intro() methods in the parent class (Fruit):

Example

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the
color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public $weight;
  public function __construct($name, $color,
$weight) {
    $this->name = $name;
    $this->color = $color;
    $this->weight = $weight;
  }
  public function intro() {
    echo "The fruit is {$this->name}, the color
is {$this->color}, and the weight is
{$this->weight} gram.";
  }
}

$strawberry = new Strawberry("Strawberry", "red",
50);
$strawberry->intro();
?>
```

PHP - The final Keyword

The **final** keyword can be used to prevent class inheritance or to prevent method overriding.

The following example shows how to prevent class inheritance:

Example

```php
<?php
final class Fruit {
  // some code
}

// will result in error
class Strawberry extends Fruit {
  // some code
}
?>
```

The following example shows how to prevent method overriding:

Example

```php
<?php
class Fruit {
  final public function intro() {
    // some code
  }
}

class Strawberry extends Fruit {
  // will result in error
  public function intro() {
    // some code
  }
}
?>
```

PHP OOP - Class Constants

PHP - Class Constants

Constants cannot be changed once it is declared.

Class constants can be useful if you need to define some constant data within a class.

A class constant is declared inside a class with the **const** keyword.

Class constants are case-sensitive. However, it is recommended to name the constants in all uppercase letters.

We can access a constant from outside the class by using the class name followed by the scope resolution operator (::) followed by the constant name, like here:

Example

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting
W3Schools.com!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

Or, we can access a constant from inside the class by using the **self** keyword followed by the scope resolution operator (::) followed by the constant name, like here:

Example

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting
W3Schools.com!";
  public function byebye() {
    echo self::LEAVING_MESSAGE;
  }
}

$goodbye = new Goodbye();
$goodbye->byebye();
?>
```

PHP OOP - Abstract Classes

PHP - What are Abstract Classes and Methods?

Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.

An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.

An abstract class or method is defined with the **abstract** keyword:

Syntax

```
<?php
abstract class ParentClass {
  abstract public function someMethod1();
  abstract public function someMethod2($name,
$color);
  abstract public function someMethod3() :
string;
}
?>
```

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

- The child class method must be defined with the same name and it redeclares the parent abstract method
- The child class method must be defined with the same or a less restricted access modifier
- The number of required arguments must be the same. However, the child class may have optional arguments in addition

Let's look at an example:

Example

```php
<?php
// Parent class
abstract class Car {
  public $name;
  public function __construct($name) {
    $this->name = $name;
  }
  abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
  public function intro() : string {
    return "Choose German quality! I'm an
$this->name!";
  }
}

class Volvo extends Car {
  public function intro() : string {
    return "Proud to be Swedish! I'm a
$this->name!";
  }
}

class Citroen extends Car {
  public function intro() : string {
    return "French extravagance! I'm a
$this->name!";
  }
}

// Create objects from the child classes
$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new volvo("Volvo");
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();
?>
```

Example Explained

The Audi, Volvo, and Citroen classes are inherited from the Car class. This means that the Audi, Volvo, and Citroen classes can use the public $name property as well as the public __construct() method from the Car class because of inheritance.

But, intro() is an abstract method that should be defined in all the child classes and they should return a string.

PHP - More Abstract Class Examples

Let's look at another example where the abstract method has an argument:

Example

```php
<?php
abstract class ParentClass {
  // Abstract method with an argument
  abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
  public function prefixName($name) {
    if ($name == "John Doe") {
      $prefix = "Mr.";
    } elseif ($name == "Jane Doe") {
      $prefix = "Mrs.";
    } else {
      $prefix = "";
    }
    return "{$prefix} {$name}";
  }
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
```

Let's look at another example where the abstract method has an argument, and the child class has two optional arguments that are not defined in the parent's abstract method:

Example

```
<?php
abstract class ParentClass {
  // Abstract method with an argument
  abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
  // The child class may define optional
arguments that are not in the parent's abstract
method
  public function prefixName($name, $separator =
".", $greet = "Dear") {
    if ($name == "John Doe") {
      $prefix = "Mr";
    } elseif ($name == "Jane Doe") {
      $prefix = "Mrs";
    } else {
      $prefix = "";
    }
    return "{$greet} {$prefix}{$separator}
{$name}";
  }
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
```

PHP OOP - Traits

PHP - What are Traits?

PHP only supports single inheritance: a child class can inherit only from one single parent.

So, what if a class needs to inherit multiple behaviors? OOP traits solve this problem.

Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).

Traits are declared with the **trait** keyword:

Syntax

```
<?php
trait TraitName {
  // some code...
}
?>
```

To use a trait in a class, use the **use** keyword:

Syntax

```
<?php
class MyClass {
  use TraitName;
}
?>
```

Let's look at an example:

Example

```
<?php
trait message1 {
public function msg1() {
    echo "OOP is fun! ";
  }
}

class Welcome {
  use message1;
}

$obj = new Welcome();
$obj->msg1();
?>
```

Example Explained

Here, we declare one trait: message1. Then, we create a class: Welcome. The class uses the trait, and all the methods in the trait will be available in the class.

If other classes need to use the msg1() function, simply use the message1 trait in those classes. This reduces code duplication, because there is no need to redeclare the same method over and over again.

PHP - Using Multiple Traits

Let's look at another example:

Example

```php
<?php
trait message1 {
  public function msg1() {
    echo "OOP is fun! ";
  }
}

trait message2 {
  public function msg2() {
    echo "OOP reduces code duplication!";
  }
}

class Welcome {
  use message1;
}

class Welcome2 {
  use message1, message2;
}

$obj = new Welcome();
$obj->msg1();
echo "<br>";

$obj2 = new Welcome2();
$obj2->msg1();
$obj2->msg2();
?>
```

Example Explained

Here, we declare two traits: message1 and message2. Then, we create two classes: Welcome and Welcome2. The first class (Welcome) uses the message1 trait, and the second class (Welcome2) uses both message1 and message2 traits (multiple traits are separated by comma).

PHP OOP - Static Methods

PHP - Static Methods

Static methods can be called directly - without creating an instance of a class.

Static methods are declared with the **static** keyword:

Syntax

```php
<?php
class ClassName {
  public static function staticMethod() {
    echo "Hello World!";
  }
}
?>
```

To access a static method use the class name, double colon (::), and the method name:

Syntax

```
ClassName::staticMethod();
```

Let's look at an example:

Example

```php
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}

// Call static method
greeting::welcome();
?>
```

Example Explained

Here, we declare a static method: welcome(). Then, we call the static method by using the class name, double colon (::), and the method name (without creating a class first).

PHP - More on Static Methods

A class can have both static and non-static methods. A static method can be accessed from a method in the same class using the self keyword and double colon (::):

Example

```
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }

  public function __construct() {
    self::welcome();
  }
}

new greeting();
?>
```

Static methods can also be called from methods in other classes. To do this, the static method should be **public**:

Example

```
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}

class SomeOtherClass {
  public function message() {
    greeting::welcome();
  }
}
?>
```

To call a static method from a child class, use the **parent** keyword inside the child class. Here, the static method can be **public** or **protected**.

Example

```
<?php
class domain {
  protected static function getWebsiteName() {
    return "W3Schools.com";
  }
}

class domainW3 extends domain {
  public $websiteName;
  public function __construct() {
    $this->websiteName =
parent::getWebsiteName();
  }
}

$domainW3 = new domainW3;
echo $domainW3 -> websiteName;
?>
```

PHP OOP - Static Properties

PHP - Static Properties

Static properties can be called directly - without creating an instance of a class.

Static properties are declared with the **static** keyword:

Syntax

```
<?php
class ClassName {
  public static $staticProp = "W3Schools";
}
?>
```

To access a static property use the class name, double colon (::), and the property name:

Syntax

```
ClassName::staticProp;
```

Let's look at an example:

Example

```php
<?php
class pi {
  public static $value = 3.14159;
}

// Get static property
echo pi::$value;
?>
```

Example Explained

Here, we declare a static property: $value. Then, we echo the value of the static property by using the class name, double colon (::), and the property name (without creating a class first).

PHP - More on Static Properties

A class can have both static and non-static properties. A static property can be accessed from a method in the same class using the self keyword and double colon (::):

Example

```php
<?php
class pi {
  public static $value=3.14159;
  public function staticValue() {
    return self::$value;
  }
}

$pi = new pi();
echo $pi->staticValue();
?>
```

To call a static property from a child class, use the **parent** keyword inside the child class:

Example

```php
<?php
class pi {
  public static $value=3.14159;
}

class x extends pi {
  public function xStatic() {
    return parent::$value;
  }
}

// Get value of static property directly via
child class
echo x::$value;

// or get value of static property via xStatic()
method
$x = new x();
echo $x->xStatic();
?>
```

PHP MySQL Database

With PHP, you can connect to and manipulate databases.

MySQL is the most popular database system used with PHP.

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is 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)

Database Queries

A query is a question or a request.

We can query a database for specific information and have a recordset returned.

Look at the following query (using standard SQL):

```
SELECT LastName FROM Employees
```

The query above selects all the data in the "LastName" column from the "Employees" table.

To learn more about SQL, please visit our SQL tutorial.

Download MySQL Database

If you don't have a PHP server with a MySQL Database, you can download it for free here: http://www.mysql.com

Facts About MySQL Database

MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia).

Another great thing about MySQL is that it can be scaled down to support embedded database applications.

Look at http://www.mysql.com/customers/ for an overview of companies using MySQL.

PHP 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)**

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

Should I Use MySQLi or PDO?

If you need a short answer, it would be "Whatever you like".

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

Both are object-oriented, but MySQLi also offers a procedural API.

Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

MySQL Examples in Both MySQLi and PDO Syntax

In this, and in the following chapters we demonstrate three ways of working with PHP and MySQL:

- MySQLi (object-oriented)
- MySQLi (procedural)
- PDO

MySQLi Installation

For Linux and Windows: The MySQLi extension is automatically installed in most cases, when php5 mysql package is installed.

For installation details, go to:
http://php.net/manual/en/mysqli.installation.php

PDO Installation

For installation details, go to:
http://php.net/manual/en/pdo.installation.php

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";
?>
```

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:

MySQLi Object-Oriented:

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

MySQLi Procedural:

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

PDO:

```
$conn = null;
```

PHP 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)

Example (MySQLi Procedural)

```php
<?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());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " .
mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Note: The following PDO example create a database named "myDBPDO":

Example (PDO)

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

try {
    $conn = new PDO("mysql:host=$servername",
$username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $sql = "CREATE DATABASE myDBPDO";
    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Database created successfully<br>";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

Tip: A great benefit of PDO is that it has 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. In the catch block above we echo the SQL statement and the generated error message.

PHP MySQL Create Table

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

Create a MySQL Table Using MySQLi and PDO

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 DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_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:

- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
- AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- 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

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
<?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 DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP
)";

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

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

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_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 DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " .
mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

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

try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

    // 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 DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP
    )";

    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Table MyGuests created successfully";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

PHP MySQL Insert Data

Insert Data Into MySQL Using MySQLi and PDO

After a database and a table have been created, we can start adding data in them.

Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table:

```
INSERT INTO table_name (column1, column2,
column3,...)
VALUES (value1, value2, value3,...)
```

To learn more about SQL, please visit our SQL tutorial.

In the previous chapter we created an empty table named "MyGuests" with five columns: "id", "firstname", "lastname", "email" and "reg_date". Now, let us fill the table with data.

Note: If a column is AUTO_INCREMENT (like the "id" column) or TIMESTAMP with default update of current_timesamp (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

The following examples add a new record to the "MyGuests" table:

Example (MySQLi Object-oriented)

```php
<?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) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" .
$conn->error;
}

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

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname,
lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" .
mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

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

try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO MyGuests (firstname,
lastname, email)
    VALUES ('John', 'Doe', 'john@example.com')";
    // use exec() because no results are returned
    $conn->exec($sql);
    echo "New record created successfully";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

PHP MySQL Get Last Inserted ID

Get ID of The 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 DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_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
<?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();
?>
```

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname,
lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
    $last_id = mysqli_insert_id($conn);
    echo "New record created successfully. Last
inserted ID is: " . $last_id;
} else {
    echo "Error: " . $sql . "<br>" .
mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

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

try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO MyGuests (firstname,
lastname, email)
    VALUES ('John', 'Doe', 'john@example.com')";
    // use exec() because no results are returned
    $conn->exec($sql);
    $last_id = $conn->lastInsertId();
    echo "New record created successfully. Last
inserted ID is: " . $last_id;
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

PHP MySQL Insert Multiple Records

Insert Multiple Records Into MySQL Using MySQLi and PDO

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
<?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.

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_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 (mysqli_multi_query($conn, $sql)) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" .
mysqli_error($conn);
}

mysqli_close($conn);
?>
```

The PDO way is a little bit different:

Example (PDO)

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

try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

    // begin the transaction
    $conn->beginTransaction();
    // our SQL statements
    $conn->exec("INSERT INTO MyGuests (firstname,
lastname, email)
    VALUES ('John', 'Doe', 'john@example.com')");
    $conn->exec("INSERT INTO MyGuests (firstname,
lastname, email)
    VALUES ('Mary', 'Moe', 'mary@example.com')");
    $conn->exec("INSERT INTO MyGuests (firstname,
lastname, email)
    VALUES ('Julie', 'Dooley',
'julie@example.com')");

    // commit the transaction
    $conn->commit();
    echo "New records created successfully";
    }
catch(PDOException $e)
    {
    // roll back the transaction if something
failed
    $conn->rollback();
    echo "Error: " . $e->getMessage();
    }

$conn = null;
?>
```

PHP MySQL 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
<?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.

Prepared Statements in PDO

The following example uses prepared statements and bound parameters in PDO:

Example (PDO with Prepared Statements)

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

try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

    // prepare sql and bind parameters
    $stmt = $conn->prepare("INSERT INTO MyGuests
(firstname, lastname, email)
    VALUES (:firstname, :lastname, :email)");
    $stmt->bindParam(':firstname', $firstname);
    $stmt->bindParam(':lastname', $lastname);
    $stmt->bindParam(':email', $email);

    // insert a row
    $firstname = "John";
    $lastname = "Doe";
    $email = "john@example.com";
    $stmt->execute();

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

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

    echo "New records created successfully";
    }
catch(PDOException $e)
    {
    echo "Error: " . $e->getMessage();
    }
$conn = null;
?>
```

PHP MySQL Select Data

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
```

To learn more about SQL, please visit our SQL tutorial.

Select Data With MySQLi

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

Example (MySQLi Object-oriented)

```php
<?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 "id: " . $row["id"]. " - Name: " .
$row["firstname"]. " " . $row["lastname"].
"<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>
```

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.

The following example shows the same as the example above, in the MySQLi procedural way:

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}

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

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

mysqli_close($conn);
?>
```

You can also put the result in an HTML table:

Example (MySQLi Object-oriented)

```php
<?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();
?>
```

Select Data With PDO (+ Prepared Statements)

The following example uses prepared statements.

It selects the id, firstname and lastname columns from the MyGuests table and displays it in an HTML table:

Example (PDO)

```php
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th>
<th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator
{
    function __construct($it) {
        parent::__construct($it,
self::LEAVES_ONLY);
    }

    function current() {
        return "<td style='width:150px;border:1px
solid black;'>" . parent::current(). "</td>";
    }

    function beginChildren() {
        echo "<tr>";
    }

    function endChildren() {
        echo "</tr>" . "\n";
    }
}

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

try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT id, firstname,
lastname FROM MyGuests");
    $stmt->execute();

    // set the resulting array to associative
    $result =
$stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new TableRows(new
RecursiveArrayIterator($stmt->fetchAll())) as
$k=>$v) {
        echo $v;
    }
}
catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

PHP MySQL Use The WHERE Clause

Select and Filter Data From a MySQL Database

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

```
SELECT column_name(s) FROM table_name WHERE
column_name operator value
```

To learn more about SQL, please visit our SQL tutorial.

Select and Filter Data With MySQLi

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

Example (MySQLi Object-oriented)

```php
<?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 WHERE lastname='Doe'";
$result = $conn->query($sql);

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

Code lines to explain from the example above:

First, we set up the SQL query that selects the id, firstname and lastname columns from the MyGuests table where the lastname is "Doe". 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.

The following example shows the same as the example above, in the MySQLi procedural way:

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM
MyGuests WHERE lastname='Doe'";
$result = mysqli_query($conn, $sql);

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

mysqli_close($conn);
?>
```

You can also put the result in an HTML table:

Example (MySQLi Object-oriented)

```php
<?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 WHERE lastname='Doe'";
$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();
?>
```

Example (PDO)

```php
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th>
<th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator
{
    function __construct($it) {
        parent::__construct($it,
self::LEAVES_ONLY);
    }

    function current() {
        return "<td style='width:150px;border:1px
solid black;'>" . parent::current(). "</td>";
    }

    function beginChildren() {
        echo "<tr>";
    }

    function endChildren() {
        echo "</tr>" . "\n";
    }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
```

Select Data With PDO (+ Prepared Statements)

The following example uses prepared statements.

It selects the id, firstname and lastname columns from the MyGuests table where the lastname is "Doe", and displays it in an HTML table:

```
try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT id, firstname,
lastname FROM MyGuests WHERE lastname='Doe'");
    $stmt->execute();

    // set the resulting array to associative
    $result =
$stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new TableRows(new
RecursiveArrayIterator($stmt->fetchAll())) as
$k=>$v) {
        echo $v;
    }
}
catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

PHP MySQL Use The ORDER BY Clause

Select and Order Data From a MySQL Database

The ORDER BY clause is used to sort the result-set in ascending or descending order.

The ORDER BY clause sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT column_name(s) FROM table_name ORDER BY
column_name(s) ASC|DESC
```

To learn more about SQL, please visit our SQL tutorial.

Select and Order Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table. The records will be ordered by the lastname column:

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

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

Code lines to explain from the example above:

First, we set up the SQL query that selects the id, firstname and lastname columns from the MyGuests table. The records will be ordered by the lastname column. 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.

The following example shows the same as the example above, in the MySQLi procedural way:

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM
MyGuests ORDER BY lastname";
$result = mysqli_query($conn, $sql);

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

mysqli_close($conn);
?>
```

You can also put the result in an HTML table:

Example (MySQLi Object-oriented)

```php
<?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 ORDER BY lastname";
$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();
?>
```

Select Data With PDO (+ Prepared Statements)

The following example uses prepared statements.

Here we select the id, firstname and lastname columns from the MyGuests table. The records will be ordered by the lastname column, and it will be displayed in an HTML table:

Example (PDO)

```
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th>
<th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator
{
    function __construct($it) {
        parent::__construct($it,
self::LEAVES_ONLY);
    }

    function current() {
        return "<td style='width:150px;border:1px
solid black;'>" . parent::current(). "</td>";
    }

    function beginChildren() {
        echo "<tr>";
    }

    function endChildren() {
        echo "</tr>" . "\n";
    }
}

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

try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT id, firstname,
lastname FROM MyGuests ORDER BY lastname");
    $stmt->execute();

    // set the resulting array to associative
    $result =
$stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new TableRows(new
RecursiveArrayIterator($stmt->fetchAll())) as
$k=>$v) {
        echo $v;
    }
}
catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

PHP MySQL Delete Data

Delete Data From a MySQL Table Using MySQLi and PDO

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!

To learn more about SQL, please visit our SQL tutorial.

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
<?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();
?>
```

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}

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

if (mysqli_query($conn, $sql)) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " .
mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

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

try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

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

    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Record deleted successfully";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

After the record is deleted, the table will look like this:

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

PHP MySQL Update Data

Update Data In a MySQL Table Using MySQLi and PDO

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!

To learn more about SQL, please visit our SQL tutorial.

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
<?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();
?>
```

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username,
$password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}

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

if (mysqli_query($conn, $sql)) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " .
mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

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

try {
    $conn = new PDO("mysql:host=$servername;
dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

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

    // Prepare statement
    $stmt = $conn->prepare($sql);

    // execute the query
    $stmt->execute();

    // echo a message to say the UPDATE succeeded
    echo $stmt->rowCount() . " records UPDATED
successfully";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

After the record is updated, the table will look like this:

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

PHP MySQL Limit Data Selections

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.

PHP XML Parsers

What is XML?

The XML language is a way to structure data for sharing across websites.

Several web technologies like RSS Feeds and Podcasts are written in XML.

XML is easy to create. It looks a lot like HTML, except that you make up your own tags.

If you want to learn more about XML, please visit our XML tutorial.

What is an XML Parser?

To read and update, create and manipulate an XML document, you will need an XML parser.

In PHP there are two major types of XML parsers:

- Tree-Based Parsers
- Event-Based Parsers

Tree-Based Parsers

Tree-based parsers holds the entire document in Memory and transforms the XML document into a Tree structure. It analyzes the whole document, and provides access to the Tree elements (DOM).

This type of parser is a better option for smaller XML documents, but not for large XML document as it causes major performance issues.

Example of tree-based parsers:

- SimpleXML
- DOM

Event-Based Parsers

Event-based parsers do not hold the entire document in Memory, instead, they read in one node at a time and allow you to interact with in real time. Once you move onto the next node, the old one is thrown away.

This type of parser is well suited for large XML documents. It parses faster and consumes less memory.

Example of event-based parsers:

- XMLReader
- XML Expat Parser

PHP SimpleXML Parser

SimpleXML is a PHP extension that allows us to easily manipulate and get XML data.

The SimpleXML Parser

SimpleXML is a tree-based parser.

SimpleXML provides an easy way of getting an element's name, attributes and textual content if you know the XML document's structure or layout.

SimpleXML turns an XML document into a data structure you can iterate through like a collection of arrays and objects.

Compared to DOM or the Expat parser, SimpleXML takes a fewer lines of code to read text data from an element.

Installation

From PHP 5, the SimpleXML functions are part of the PHP core. No installation is required to use these functions.

PHP SimpleXML - Read From String

The PHP **simplexml_load_string()** function is used to read XML data from a string.

Assume we have a variable that contains XML data, like this:

```
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>";
```

The example below shows how to use the **simplexml_load_string()** function to read XML data from a string:

Example

```php
<?php
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>";

$xml=simplexml_load_string($myXMLData) or
die("Error: Cannot create object");
print_r($xml);
?>
```

The output of the code above will be:

```
SimpleXMLElement Object ( [to] => Tove [from]
=> Jani [heading] => Reminder [body] => Don't
forget me this weekend! )
```

Error Handling Tip: Use the libxml functionality to retrieve all XML errors when loading the document and then iterate over the errors. The following example tries to load a broken XML string:

Example

```php
<?php
libxml_use_internal_errors(true);
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<document>
<user>John Doe</wronguser>
<email>john@example.com</wrongemail>
</document>";

$xml = simplexml_load_string($myXMLData);
if ($xml === false) {
    echo "Failed loading XML: ";
    foreach(libxml_get_errors() as $error) {
        echo "<br>", $error->message;
    }
} else {
    print_r($xml);
}
?>
```

The output of the code above will be:

```
Failed loading XML:
Opening and ending tag mismatch: user line 3
and wronguser
Opening and ending tag mismatch: email line 4
and wrongemail
```

PHP SimpleXML - Read From File

The PHP **simplexml_load_file()** function is used to read XML data from a file.

Assume we have an XML file called "note.xml", that looks like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The example below shows how to use the **simplexml_load_file()** function to read XML data from a file:

Example

```php
<?php
$xml=simplexml_load_file("note.xml") or
die("Error: Cannot create object");
print_r($xml);
?>
```

The output of the code above will be:

```
SimpleXMLElement Object ( [to] => Tove [from]
=> Jani [heading] => Reminder [body] => Don't
forget me this weekend! )
```

Tip: The next chapter shows how to get/retrieve node values from an XML file with SimpleXML!

More PHP SimpleXML

For more information about the PHP SimpleXML functions, visit our PHP SimpleXML Reference.

PHP SimpleXML - Get Node/Attribute Values

SimpleXML is a PHP extension that allows us to easily manipulate and get XML data.

PHP SimpleXML - Get Node Values

Get the node values from the "note.xml" file:

Example

```php
<?php
$xml=simplexml_load_file("note.xml") or
die("Error: Cannot create object");
echo $xml->to . "<br>";
echo $xml->from . "<br>";
echo $xml->heading . "<br>";
echo $xml->body;
?>
```

The output of the code above will be:

```
Tove
Jani
Reminder
Don't forget me this weekend!
```

Another XML File

Assume we have an XML file called "books.xml", that looks like this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en-us">XQuery Kick
Start</title>
    <author>James McGovern</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
  <book category="WEB">
    <title lang="en-us">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

PHP SimpleXML - Get Node Values of Specific Elements

The following example gets the node value of the <title> element in the first and second <book> elements in the "books.xml" file:

Example

```php
<?php
$xml=simplexml_load_file("books.xml") or
die("Error: Cannot create object");
echo $xml->book[0]->title . "<br>";
echo $xml->book[1]->title;
?>
```

The output of the code above will be:

```
Everyday Italian
Harry Potter
```

PHP SimpleXML - Get Node Values - Loop

The following example loops through all the <book> elements in the "books.xml" file, and gets the node values of the <title>, <author>, <year>, and <price> elements:

Example

```
<?php
$xml=simplexml_load_file("books.xml") or
die("Error: Cannot create object");
foreach($xml->children() as $books) {
    echo $books->title . ", ";
    echo $books->author . ", ";
    echo $books->year . ", ";
    echo $books->price . "<br>";
}
?>
```

The output of the code above will be:

```
Everyday Italian, Giada De Laurentiis, 2005,
30.00
Harry Potter, J K. Rowling, 2005, 29.99
XQuery Kick Start, James McGovern, 2003, 49.99
Learning XML, Erik T. Ray, 2003, 39.95
```

PHP SimpleXML - Get Attribute Values

The following example gets the attribute value of the "category" attribute of the first <book> element and the attribute value of the "lang" attribute of the <title> element in the second <book> element:

Example

```
<?php
$xml=simplexml_load_file("books.xml") or
die("Error: Cannot create object");
echo $xml->book[0]['category'] . "<br>";
echo $xml->book[1]->title['lang'];
?>
```

The output of the code above will be:

```
COOKING
en
```

PHP SimpleXML - Get Attribute Values - Loop

The following example gets the attribute values of the <title> elements in the "books.xml" file:

Example

```
<?php
$xml=simplexml_load_file("books.xml") or
die("Error: Cannot create object");
foreach($xml->children() as $books) {
    echo $books->title['lang'];
    echo "<br>";
}
?>
```

The output of the code above will be:

```
en
en
en-us
en-us
```

More PHP SimpleXML

For more information about the PHP SimpleXML functions, visit our PHP SimpleXML Reference.

PHP XML Expat Parser

The built-in XML Expat Parser makes it possible to process XML documents in PHP.

The XML Expat Parser

The Expat parser is an event-based parser.

Look at the following XML fraction:

```
<from>Jani</from>
```

An event-based parser reports the XML above as a series of three events:

- Start element: from
- Start CDATA section, value: Jani
- Close element: from

The XML Expat Parser functions are part of the PHP core. There is no installation needed to use these functions.

The XML File

The XML file "note.xml" will be used in the example below:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Initializing the XML Expat Parser

We want to initialize the XML Expat Parser in PHP, define some handlers for different XML events, and then parse the XML file.

Example

```
<?php
// Initialize the XML parser
$parser=xml_parser_create();

// Function to use at the start of an element
function
start($parser,$element_name,$element_attrs) {
  switch($element_name) {
    case "NOTE":
    echo "-- Note --<br>";
    break;
    case "TO":
    echo "To: ";
    break;
    case "FROM":
    echo "From: ";
    break;
    case "HEADING":
    echo "Heading: ";
    break;
    case "BODY":
    echo "Message: ";
  }
}

// Function to use at the end of an element
function stop($parser,$element_name) {
  echo "<br>";
}

// Function to use when finding character data
function char($parser,$data) {
  echo $data;
}
```

```
// Specify element handler
xml_set_element_handler($parser,"start","stop");

// Specify data handler
xml_set_character_data_handler($parser,"char");

// Open XML file
$fp=fopen("note.xml","r");

// Read data
while ($data=fread($fp,4096)) {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
}

// Free the XML parser
xml_parser_free($parser);
?>
```

Example explained:

1. Initialize the XML parser with the **xml_parser_create()** function
2. Create functions to use with the different event handlers
3. Add the **xml_set_element_handler()** function to specify which function will be executed when the parser encounters the opening and closing tags
4. Add the **xml_set_character_data_handler()** function to specify which function will execute when the parser encounters character data
5. Parse the file "note.xml" with the **xml_parse()** function
6. In case of an error, add **xml_error_string()** function to convert an XML error to a textual description
7. Call the **xml_parser_free()** function to release the memory allocated with the **xml_parser_create()** function

More PHP XML Expat Parser

For more information about the PHP Expat functions, visit our PHP XML Parser Reference.

PHP XML DOM Parser

The built-in DOM parser makes it possible to process XML documents in PHP.

The XML DOM Parser

The DOM parser is a tree-based parser.

Look at the following XML document fraction:

```
<?xml version="1.0" encoding="UTF-8"?>
<from>Jani</from>
```

The DOM sees the XML above as a tree structure:

- Level 1: XML Document
- Level 2: Root element: <from>
- Level 3: Text element: "Jani"

Installation

The DOM parser functions are part of the PHP core. There is no installation needed to use these functions.

The XML File

The XML file below ("note.xml") will be used in our example:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Load and Output XML

We want to initialize the XML parser, load the xml, and output it:

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
?>
```

The output of the code above will be:

```
Tove Jani Reminder Don't forget me this
weekend!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The example above creates a DOMDocument-Object and loads the XML from "note.xml" into it.

Then the saveXML() function puts the internal XML document into a string, so we can output it.

Looping through XML

We want to initialize the XML parser, load the XML, and loop through all elements of the <note> element:

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item) {
  print $item->nodeName . " = " .
$item->nodeValue . "<br>";
}
?>
```

The output of the code above will be:

```
#text =
to = Tove
#text =
from = Jani
#text =
heading = Reminder
#text =
body = Don't forget me this weekend!
#text =
```

In the example above you see that there are empty text nodes between each element.

When XML generates, it often contains white-spaces between the nodes. The XML DOM parser treats these as ordinary elements, and if you are not aware of them, they sometimes cause problems.

If you want to learn more about the XML DOM, please visit our XML tutorial.

PHP - AJAX Introduction

AJAX is about updating parts of a web page, without reloading the whole page.

What is AJAX?

AJAX = Asynchronous JavaScript and XML.

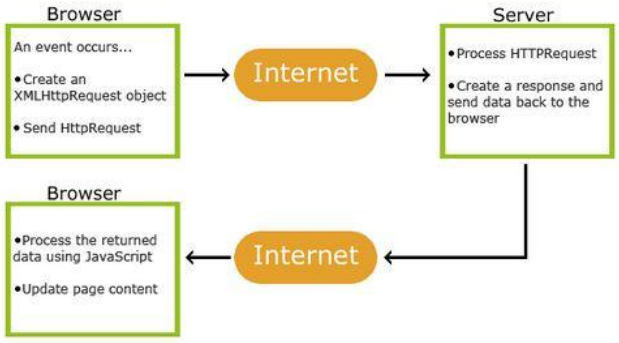
AJAX is a technique for creating fast and dynamic web pages.

AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.

Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.

How AJAX Works



AJAX is Based on Internet Standards

AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

AJAX applications are browser- and platform-independent!

Google Suggest

AJAX was made popular in 2005 by Google, with Google Suggest.

Google Suggest is using AJAX to create a very dynamic web interface: When you start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions.

Start Using AJAX Today

In our PHP tutorial, we will demonstrate how AJAX can update parts of a web page, without reloading the whole page. The server script will be written in PHP.

If you want to learn more about AJAX, visit our AJAX tutorial.

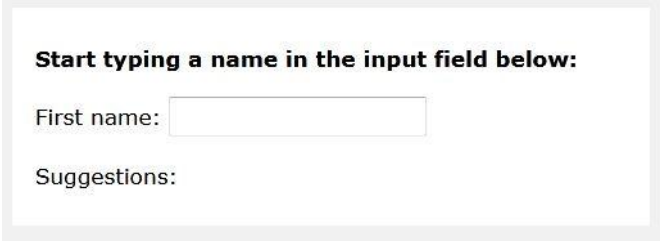
PHP - AJAX and PHP

AJAX is used to create more interactive applications.

AJAX PHP Example

The following example will demonstrate how a web page can communicate with a web server while a user type characters in an input field:

Example



Example Explained

In the example above, when a user types a character in the input field, a function called "showHint()" is executed.

The function is triggered by the onkeyup event.

Here is the HTML code:

Example

```
<html>
<head>
<script>
function showHint(str) {
    if (str.length == 0) {
        document.getElementById("txtHint").innerHTML = "";
        return;
    } else {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200)
{

document.getElementById("txtHint").innerHTML =
this.responseText;
            }
        };
        xmlhttp.open("GET", "gethint.php?q=" + str, true);
        xmlhttp.send();
    }
}
</script>
</head>
<body>

<p><b>Start typing a name in the input field below:</b></p>
<form>
First name: <input type="text"
onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

Code explanation:

First, check if the input field is empty (str.length == 0). If it is, clear the content of the txtHint placeholder and exit the function.

However, if the input field is not empty, do the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a PHP file (gethint.php) on the server
- Notice that q parameter is added to the url (gethint.php?q="+str)
- And the str variable holds the content of the input field

The PHP File - "gethint.php"

The PHP file checks an array of names, and returns the corresponding name(s) to the browser:

```
<?php
// Array with names
$a[] = "Anna";
$a[] = "Brittany";
$a[] = "Cinderella";
$a[] = "Diana";
$a[] = "Eva";
$a[] = "Fiona";
$a[] = "Gunda";
$a[] = "Hege";
$a[] = "Inga";
$a[] = "Johanna";
$a[] = "Kitty";
$a[] = "Linda";
$a[] = "Nina";
$a[] = "Ophelia";
$a[] = "Petunia";
$a[] = "Amanda";
$a[] = "Raquel";
$a[] = "Cindy";
$a[] = "Doris";
$a[] = "Eve";
$a[] = "Evita";
$a[] = "Sunniva";
$a[] = "Tove";
$a[] = "Unni";
$a[] = "Violet";
$a[] = "Liza";
$a[] = "Elizabeth";
$a[] = "Ellen";
$a[] = "Wenche";
$a[] = "Vicky";
```

```php
// get the q parameter from URL
$q = $_REQUEST["q"];

$hint = "";

// lookup all hints from array if $q is different
from ""
if ($q !== "") {
    $q = strtolower($q);
    $len=strlen($q);
    foreach($a as $name) {
        if (stristr($q, substr($name, 0, $len)))
{
            if ($hint === "") {
                $hint = $name;
            } else {
                $hint .= ", $name";
            }
        }
    }
}

// Output "no suggestion" if no hint was found or
output correct values
echo $hint === "" ? "no suggestion" : $hint;
?>
```

id	FirstName	LastName	Age	Hometown	Job
1	Peter	Griffin	41	Quahog	Brewery
2	Lois	Griffin	40	Newport	Piano Teacher
3	Joseph	Swanson	39	Quahog	Police Officer
4	Glenn	Quagmire	41	Quahog	Pilot

Example Explained

In the example above, when a user selects a person in the dropdown list above, a function called "showUser()" is executed.

The function is triggered by the onchange event.

Here is the HTML code:

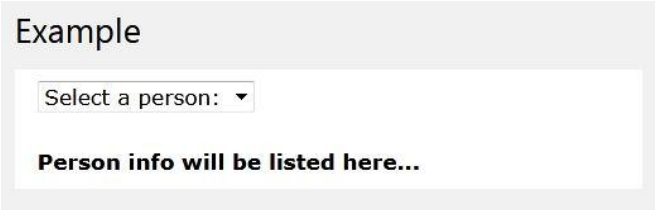
PHP - AJAX and MySQL

AJAX can be used for interactive communication with a database.

AJAX Database Example

The following example will demonstrate how a web page can fetch information from a database with AJAX:

Example



Example Explained - The MySQL Database

The database table we use in the example above looks like this:

Example

```
<html>
<head>
<script>
function showUser(str) {
    if (str == "") {

document.getElementById("txtHint").innerHTML =
"";
        return;
    } else {
        if (window.XMLHttpRequest) {
            // code for IE7+, Firefox, Chrome,
Opera, Safari
            xmlhttp = new XMLHttpRequest();
        } else {
            // code for IE6, IE5
            xmlhttp = new
ActiveXObject("Microsoft.XMLHTTP");
        }
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 &&
this.status == 200) {

document.getElementById("txtHint").innerHTML =
this.responseText;
            }
        };

xmlhttp.open("GET","getuser.php?q="+str,true);
        xmlhttp.send();
    }
}
</script>
</head>
<body>

<form>
<select name="users"
onchange="showUser(this.value)">
  <option value="">Select a person:</option>
  <option value="1">Peter Griffin</option>
  <option value="2">Lois Griffin</option>
  <option value="3">Joseph Swanson</option>
  <option value="4">Glenn Quagmire</option>
  </select>
</form>
<br>
<div id="txtHint"><b>Person info will be listed
here...</b></div>

</body>
</html>
```

Code explanation:

First, check if person is selected. If no person is selected (str == ""), clear the content of txtHint and exit the function. If a person is selected, do the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "getuser.php".

The source code in "getuser.php" runs a query against a MySQL database, and returns the result in an HTML table:

```
<!DOCTYPE html>
<html>
<head>
<style>
table {
    width: 100%;
    border-collapse: collapse;
}

table, td, th {
    border: 1px solid black;
    padding: 5px;
}

th {text-align: left;}
</style>
</head>
<body>

<?php
$q = intval($_GET['q']);

$con =
mysqli_connect('localhost','peter','abc123','my_d
b');
if (!$con) {
    die('Could not connect: ' .
mysqli_error($con));
}
mysqli_select_db($con,"ajax_demo");
$sql="SELECT * FROM user WHERE id = '".$q."'";
$result = mysqli_query($con,$sql);

echo "<table>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";
while($row = mysqli_fetch_array($result)) {
    echo "<tr>";
    echo "<td>" . $row['FirstName'] . "</td>";
    echo "<td>" . $row['LastName'] . "</td>";
    echo "<td>" . $row['Age'] . "</td>";
    echo "<td>" . $row['Hometown'] . "</td>";
    echo "<td>" . $row['Job'] . "</td>";
    echo "</tr>";
}
echo "</table>";
mysqli_close($con);
?>
</body>
</html>
```

Explanation: When the query is sent from the JavaScript to the PHP file, the following happens:

1. PHP opens a connection to a MySQL server
2. The correct person is found
3. An HTML table is created, filled with data, and sent back to the "txtHint" placeholder

PHP Example - AJAX and XML

AJAX can be used for interactive communication with an XML file.

AJAX XML Example

The following example will demonstrate how a web page can fetch information from an XML file with AJAX:

Example

Select a CD:

CD info will be listed here...

Example Explained - The HTML Page

When a user selects a CD in the dropdown list above, a function called "showCD()" is executed. The function is triggered by the "onchange" event:

```
<html>
<head>
<script>
function showCD(str) {
  if (str=="") {

document.getElementById("txtHint").innerHTML="";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera,
Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new
ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (this.readyState==4 && this.status==200) {

document.getElementById("txtHint").innerHTML=this
.responseText;
    }
  }
  xmlhttp.open("GET","getcd.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
<body>

<form>
Select a CD:
<select name="cds" onchange="showCD(this.value)">
<option value="">Select a CD:</option>
<option value="Bob Dylan">Bob Dylan</option>
<option value="Bee Gees">Bee Gees</option>
<option value="Cat Stevens">Cat Stevens</option>
</select>
</form>
<div id="txtHint"><b>CD info will be listed
here...</b></div>

</body>
</html>
```

The showCD() function does the following:

- Check if a CD is selected
- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "getcd.php".

The PHP script loads an XML document, "cd_catalog.xml", runs a query against the XML file, and returns the result as HTML:

```php
<?php
$q=$_GET["q"];

$xmlDoc = new DOMDocument();
$xmlDoc->load("cd_catalog.xml");

$x=$xmlDoc->getElementsByTagName('ARTIST');

for ($i=0; $i<=$x->length-1; $i++) {
  //Process only element nodes
  if ($x->item($i)->nodeType==1) {
    if
($x->item($i)->childNodes->item(0)->nodeValue ==
$q) {
      $y=($x->item($i)->parentNode);
    }
  }
}

$cd=($y->childNodes);

for ($i=0;$i<$cd->length;$i++) {
  //Process only element nodes
  if ($cd->item($i)->nodeType==1) {
    echo("<b>" . $cd->item($i)->nodeName . ":</b>
");

echo($cd->item($i)->childNodes->item(0)->nodeValu
e);
    echo("<br>");
  }
}
?>
```

When the CD query is sent from the JavaScript to the PHP page, the following happens:

1. PHP creates an XML DOM object
2. Find all <artist> elements that matches the name sent from the JavaScript
3. Output the album information (send to the "txtHint" placeholder)

PHP Example - AJAX Live Search

AJAX can be used to create more user-friendly and interactive searches.

AJAX Live Search

The following example will demonstrate a live search, where you get search results while you type.

Live search has many benefits compared to traditional searching:

- Results are shown as you type
- Results narrow as you continue typing
- If results become too narrow, remove characters to see a broader result

Search for a W3Schools page in the input field below:

The results in the example above are found in an XML file (links.xml). To make this example small and simple, only six results are available.

Example Explained - The HTML Page

When a user types a character in the input field above, the function "showResult()" is executed. The function is triggered by the "onkeyup" event:

```
<html>
<head>
<script>
function showResult(str) {
  if (str.length==0) {

  document.getElementById("livesearch").innerHTML
="";

document.getElementById("livesearch").style.borde
r="0px";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera,
Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new
ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (this.readyState==4 && this.status==200) {

  document.getElementById("livesearch").innerHTML
=this.responseText;

document.getElementById("livesearch").style.borde
r="1px solid #A5ACB2";
    }
  }

xmlhttp.open("GET","livesearch.php?q="+str,true);
  xmlhttp.send();
}
</script>

</head>
<body>

<form>
<input type="text" size="30"
onkeyup="showResult(this.value)">
<div id="livesearch"></div>
</form>

</body>
</html>
```

Source code explanation:

If the input field is empty (str.length==0), the function clears the content of the livesearch placeholder and exits the function.

If the input field is not empty, the showResult() function executes the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the input field)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "livesearch.php".

The source code in "livesearch.php" searches an XML file for titles matching the search string and returns the result:

```
<?php
$xmlDoc=new DOMDocument();
$xmlDoc->load("links.xml");

$x=$xmlDoc->getElementsByTagName('link');

//get the q parameter from URL
$q=$_GET["q"];

//lookup all links from the xml file if length of
q>0
if (strlen($q)>0) {
  $hint="";
  for($i=0; $i<($x->length); $i++) {

  $y=$x->item($i)->getElementsByTagName('title');
    $z=$x->item($i)->getElementsByTagName('url');
    if ($y->item(0)->nodeType==1) {
      //find a link matching the search text
      if
(stristr($y->item(0)->childNodes->item(0)->nodeVa
lue,$q)) {
        if ($hint=="") {
          $hint="<a href='" .

      $z->item(0)->childNodes->item(0)->nodeValue
.
          "' target='_blank'>" .

  $y->item(0)->childNodes->item(0)->nodeValue .
"</a>";
        } else {
          $hint=$hint . "<br /><a href='" .
```

```
      $z->item(0)->childNodes->item(0)->nodeValue
.
          "' target='_blank'>" .

  $y->item(0)->childNodes->item(0)->nodeValue .
"</a>";
        }
      }
    }
  }
}

// Set output to "no suggestion" if no hint was
found
// or to the correct values
if ($hint=="") {
  $response="no suggestion";
} else {
  $response=$hint;
}

//output the response
echo $response;
?>
```

If there is any text sent from the JavaScript (strlen($q) > 0), the following happens:

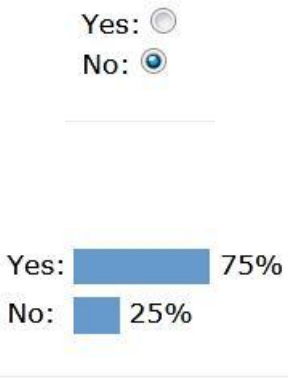
- Load an XML file into a new XML DOM object
- Loop through all <title> elements to find matches from the text sent from the JavaScript
- Sets the correct url and title in the "$response" variable. If more than one match is found, all matches are added to the variable
- If no matches are found, the $response variable is set to "no suggestion"

PHP Example - AJAX Poll

AJAX Poll

The following example will demonstrate a poll where the result is shown without reloading.

Result:



Example Explained - The HTML Page

When a user chooses an option above, a function called "getVote()" is executed. The function is triggered by the "onclick" event:

```
<html>
<head>
<script>
function getVote(int) {
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera,
Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new
ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (this.readyState==4 && this.status==200) {

  document.getElementById("poll").innerHTML=this.
responseText;
    }
  }

xmlhttp.open("GET","poll_vote.php?vote="+int,true
);
  xmlhttp.send();
}
</script>
</head>
<body>
```

```
<div id="poll">
<h3>Do you like PHP and AJAX so far?</h3>
<form>
Yes:
<input type="radio" name="vote" value="0"
onclick="getVote(this.value)">
<br>No:
<input type="radio" name="vote" value="1"
onclick="getVote(this.value)">
</form>
</div>

</body>
</html>
```

The getVote() function does the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (vote) is added to the URL (with the value of the yes or no option)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "poll_vote.php":

```
<?php
$vote = $_REQUEST['vote'];

//get content of textfile
$filename = "poll_result.txt";
$content = file($filename);

//put content in array
$array = explode("||", $content[0]);
$yes = $array[0];
$no = $array[1];

if ($vote == 0) {
  $yes = $yes + 1;
}
if ($vote == 1) {
  $no = $no + 1;
}

//insert votes to txt file
$insertvote = $yes."||".$no;
$fp = fopen($filename,"w");
fputs($fp,$insertvote);
fclose($fp);
?>


<h2>Result:</h2>
<table>
<tr>
<td>Yes:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($yes/($no+$yes),2));
?>'
height='20'>
<?php echo(100*round($yes/($no+$yes),2)); ?>%
</td>
</tr>
<tr>
<td>No:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($no/($no+$yes),2));
?>'
height='20'>
<?php echo(100*round($no/($no+$yes),2)); ?>%
</td>
</tr>
</table>
```

The value is sent from the JavaScript, and the following happens:

1. Get the content of the "poll_result.txt" file
2. Put the content of the file in variables and add one to the selected variable
3. Write the result to the "poll_result.txt" file
4. Output a graphical representation of the poll result

The Text File

The text file (poll_result.txt) is where we store the data from the poll.

It is stored like this:

```
0||0
```

The first number represents the "Yes" votes, the second number represents the "No" votes.

Note: Remember to allow your web server to edit the text file. Do **NOT** give everyone access, just the web server (PHP).