PHP Tutorial

Compiled by: Halil Özmen (parts were written by Stig Sæther Bakken //www.zend.com/zend/art/intro.php)

CONTENTS

What is PHP?

PHP (recursive acronym for "PHP: Hypertext Preprocessor") is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

How PHP came into being

PHP started as a quick Perl hack written by Rasmus Lerdorf in late 1994. Over the next two to three years, it evolved into what we today know as PHP/FI 2.0. PHP/FI started to get a lot of users, but things didn't start flying until Zeev Suraski and Andi Gutmans suddenly came along with a new parser in the summer of 1997, leading to PHP 3.0. PHP 3.0 defined the syntax and semantics used in both versions 3 and 4.

Why yet another language?

People often ask "why invent yet another language; don't we have enough of them out there"? It is simply a matter of "the right tool for the right job". Many Web developers found that existing tools and languages were not ideal for the specific task of embedding code in markup. Those developers collaborated to develop a server-side scripting language which they felt would be ideal for developing dynamic Web-based sites and applications.

PHP was created with these particular needs in mind. Moreover, PHP code was developed for embedment within HTML. In doing so, it was hoped that benefits such as quicker response time, improved security, and transparency to the end user would be achieved. PHP has evolved into a language, or maybe even an environment, that has a very specific range of tasks in mind. For this, PHP is, in Stig's humble opinion, pretty close to the ideal tool.

For More Information

The following resources will further your knowledge of PHP:

The PHP Manual:	http://www.php.net/manual/en/
The PHP FAQ:	http://tr2.php.net/faq

Language Syntax

Most of PHP's syntax is borrowed from C, although there are elements borrowed from Perl, C++ and Java as well. This article assumes that you are familiar with C's syntax. However, don't panic if you're not.

Embedding PHP Code

To give you an idea of what embedding PHP would entail, consider the following three "hello world" examples, all of which will give the exact same output:

Example 1: HTML alone

```
Hello, World!
```

Example 2: PHP code alone

```
<?php print "Hello, World!"; ?>
```

Example 3: PHP embedded within HTML

```
<?php print "Hello,"; ?> World!
```

Web servers supporting PHP will, by default, scan a file in HTML mode. HTML code will be passed over to the browser as usual, up until the server happens upon a PHP line of code. In examples 2 and 3 above, the "**<? php**" tag informs the server that PHP code is to follow. The server then switches over to PHP mode in anticipation of a PHP command. The "**?>**" tag closes out the PHP mode with the server resuming its scanning in HTML mode once more.

Embedding code in this manner is, conceptually, a more fluid approach to designing a Web page because you are working within the output setting, namely an HTML page. Traditionally, you had to fragment the output (i.e. the header, body, footer etc..) and then put it into the code. Now we are inserting the code directly into the output.

"So, what's the difference?" or "Why add extra code when HTML alone would do the trick?".

Dynamic vs. Static Web pages

The "Hello, World" example we chose would certainly not require you to use PHP. That's because it is static, meaning its display will always remain the same. But what if you wanted to greet the world in any number of ways? Say, for example, "Bonjour, World!", or "Yo, World!" and so on.

Since HTML tags are purely descriptive they cannot function as a variable. Nor can they convey even the simplest of uncertainty such as a "Bonjour" or a "Yo". You need a command language to handle variability in a Web page. Based on either a conditional statement or direct user input, a command language can generate the "static" HTML necessary to correctly display a Web page's content.

Let us reconsider example #3. This time we want to let the user decide how to greet the world:

Example 4: PHP embedded within HTML revisited!

```
<?php print $greeting, ", "; ?> World!
```

From the above example, $greeting is assigned a value, and together with the comma and the word "World!", this value is sent to the browser.

Dynamic Web page design, however, is more than just about inserting variables. What if you wanted not only to greet the world in French, but also to present the page using the colors of the French flag?

Both a Web page's structure as well as its content can be customized. This means dynamic Web page programming can also entail on-demand Web page building. No static, here!

Variables

In PHP, a variable does not require formal declaration. It will automatically be declared when a value is assigned to it. Variables are prefixed by a **dollar sign**: ($VariableName).

Variables do not have declared types. A variable's type does not have to be fixed, meaning it can be changed over the variable's lifetime. The table below list's PHP's variable types:

Type	Description
Integer	integer number
Double	floating point number
bool[1]	Boolean (true or false), available from PHP 4.0
Array	hybrid of ordered array and associative array
object[2]	an object with properties and methods (not discussed in this article)

In the following example, four variables are automatically declared by assigning a value to them:

```
<?php
$number = 5;
$string1 = "this is a string\n";
$string2 = 'this is another "string"';
$real = 37.2;
?>
```

Strings

The string constants may be enclosed between double-quotes (") or single-quotes (').
In a double-quote enclosed string, the variable conversion is done automatically.

```
$city = "Ankara";
print "I live in $city.";
$height = 182;
print "I am $height cm tall.";
```

String operators: Concatenation: **.** Concatenation to the end: **.=**

```
$str1 = $lastname . ", " . $firstname;
$str2 .=  $str3;
```

Arrays

PHP supports two type of arrays:

- Numerically indexed arrays
- Associative arrays.

Numerically indexed arrays:

The indices in numerically indexed arrays starts from 0 by default, although this can be altered.

The elements of an array can be of various data types. Integers, doubles and strings can be used in the same array.

There is no declaration of size. All arrays are dynamic in PHP.

Creation of arrays:

```
$ar1 = array ();   // here an empty array is created.
$ar2 = array (7, 12, 20, 32);    // an array with indices 0 to 3
$ar3 = array (12, "January", 2008, 16.40); // array with various data types
$ar4 = array(5 => "five", "six", "seven");  // an array with indices 5 to 7
```

Accessing array contents:

```
print $ar2[2];  // 20 is output
$ar3[2] = 2009;     // value of the element with index 2 is changed
$ar1[0] = "Adana";  // an element is added to $ar1
$ar3[4] = "East";   // an element is added to $ar3
printf("7: %d, 1: %d, 'six': %s\n", $array1[3], $array2["one"], $array3[6]);
foreach ($ar3 as $v)
{  print  "$v<br />";  }         // all values of the array are output
```

Associative Arrays

Associative arrays has two part: keys and values. The keys are indices of the array.

Both the keys and values can be of any data type in the same array.

=> operator is used to combine the key and the value of an element of an associative array.

Creation of associative arrays:

```
$as1 = array(24 => "Ali", "Ankara" => 844, "Pi" => 3.14159);
$as2 = array(20458444 => "Ali Ak", 20487542 => "Ayse Bal");
```

keys	values
24	"Ali"
"Ankara"	844
"Pi"	3.14159

Accessing array contents:

```
print $as1["Ankara"];  // outputs: 844
$as1[24] = "Ayse";     // value of element with key=24 is changed.
$as2[20608888] = "Fatma Girik";  // a new element is added to array.
foreach ($as2 as $id => $name)
{  print  "<b>$id</b>: $name<br />";  }    // all keys and values are used
foreach ($as4 as $k => $v)
{  $as4[$k] = $v + 7;  }     // all values of array $as4 are increased by 7
```

Conditionals and Looping Constructs

PHP includes if and elseif conditionals, as well as while and for loops, all with syntax similar to C. The example below introduces these four constructs:

```
// Conditionals
if ($a)
{
    print "a is true<br />\n";
}
else if ($b)
{
    print "b is true<br />\n";
}
else
{
    print "neither a or b is true<br />\n";
}

// Loops
for ($j = 0; $j < 10; $j++)
{
    print "j = $j<br />\n";
}

while ($d)
{
    print "ok<br />\n";
    $d = test_something();
}

do
{
    $c = test_something();
} while ($c);
```

Array Traverse Constructs:

foreach ($array as $value) {...} // To traverse elements (values) in an array.
{ .... }

foreach ($array as $key => $value) {...} // To traverse elements (key + value) in an associative array.
{ .... }

Operators

Assignment operators: = += -= *= /= .= (string concatenation)

Arithmetic Operators

Example	Name	Result
$a + $b	Addition	Sum of $a and $b.
$a - $b	Subtraction	Difference of $a and $b.
$a * $b	Multiplication	Product of $a and $b.
$a / $b	Division	Quotient of $a and $b.
$a % $b	Modulus	Remainder of $a divided by $b.

Incrementing/Decrementing Operators

Example	Name	Effect
++$a	Pre-increment	Increments $a by one, then returns $a.
$a++	Post-increment	Returns $a, then increments $a by one.
--$a	Pre-decrement	Decrements $a by one, then returns $a.
$a--	Post-decrement	Returns $a, then decrements $a by one.

Comparison Operators

Example	Name	Result
$a == $b	Equal	**TRUE** if $a is equal to $b.
$a != $b	Not equal	**TRUE** if $a is not equal to $b.
$a <> $b	Not equal	**TRUE** if $a is not equal to $b.
$a < $b	Less than	**TRUE** if $a is strictly less than $b.
$a <= $b	Less than or equal to	**TRUE** if $a is less than or equal to $b.
$a > $b	Greater than	**TRUE** if $a is strictly greater than $b.
$a >= $b	Greater than or equal to	**TRUE** if $a is greater than or equal to $b.
$a === $b	Identical	**TRUE** if $a is equal to $b, and they are of the same type. (PHP 4 only)
$a !== $b	Not identical	**TRUE** if $a is not equal to $b, or they are not of the same type. (PHP 4 only)

Logical Operators

Example	Name	Result
$a && $b	And	**TRUE** if both $a and $b are **TRUE**.
$a and $b	And	**TRUE** if both $a and $b are **TRUE**.
$a || $b	Or	**TRUE** if either $a or $b is **TRUE**.
$a or $b	Or	**TRUE** if either $a or $b is **TRUE**.
! $a	Not	**TRUE** if $a is not **TRUE**.
$a xor $b	Xor	**TRUE** if either $a or $b is **TRUE**, but not both.

String operators: Concatenation: **.** Concatenation to the end: **.=**

```
$str1 = $lastname . ", " . $firstname;
$str2 .=  $str3;
```

Functions

Function declaration:

```
function  my_func ()
{
  print "Hello Ankara!";
}  // end function  my_func
```

Calling the function:

```
my_func();
```

Function parameters and returning values:

```
function  count_a ($str)
{
  $n = substr_count($str, "a");;
  return ($n);
}  // end function  count_a
```

Calling the function:

```
$x = count_a ($name);
```

Function parameters - passing parameters **by reference** and **by value**:

```
function  salary_increase (&$value, $inc_rate)
{
  $value *= (1 + $inc_rate);
}  // end function  salary_increase
```

Calling the function:

```
salary_increase ($salary, 0.1);
```

Returning arrays:

```
function  get_values ($a, $b)
{
  $vals = array();
  // ...
  // statements to put values into the array $vals
  // ...
  return ($vals);
}  // end function  get_values
```

Calling the function:

```
$xarr = get_values ($k, $m);
```

// Here, $xarr becomes an array

File System Functions:

Open File	$fp = fopen(“filename”, “mode”);	Open a file, where mode is either “r”, “w” or “a”
Close File	fclose ($fp);	
Read a line from file	$line = fgets($fp, 4096);	
Write a string to a file	fwrite ($fp, $string);	

Read all the lines of a file	$line = fgets($fp, 4096);
	while(!feof($fp))
	{ print "$line
	\n";
	$line = fgets($fp, 4096);
	}
		Reads line by line while NOT end-of-file.

Read all the lines of a file	$lines = file ('filename'); foreach ($lines as $line) { print $line; }	Reads all the lines of the file in an array; then print the lines.
Delete a file	unlink ('filename');	

See **http://www.php.net/manual/en/ref.filesystem.php** for more information.

Web Application Features

One of PHP's oldest features is the ability to make HTML form and cookie data available directly to the programmer. By default, any form entry creates a global PHP variable of the same name.

In the following example, a user name is retrieved and assigned to a variable. The name is then printed by the sub-routine "submit.php":

```
<FORM  ACTION="submit.php"  METHOD="post">
What's your name? <INPUT NAME="myname" SIZE=3>
<br /> <input type="submit" value=" SUBMIT " name="btn_submit">
</FORM>
```

submit.php

```
<?php
print "Hello, $myname!";
?>
```

Working With Cookies

HTTP cookies (or just cookies) are parcels of text sent by a server to a Web client (usually a browser) and then sent back unchanged by the client each time it accesses that server.

Each cookie has a name and value. It may also have an expiration time.

Cookies are sent by web server inside the HTTP response before any html content.

Cookies are stored in the client computer.

In PHP, to create cookie, the function setcookie is used. Example:

```
setcookie ("ck_cnt", $count, time()+3600*24*365);  // expires in a year
```

In PHP, the cookies can be accessed using the super global variable $_COOKIE.

The PHP statement: `extract ($_COOKIE);`

creates variables with the names of cookies containing their respective values.

HTTP cookies are used for authenticating, session tracking (state maintenance), and maintaining specific information about users, such as site preferences or the contents of their electronic shopping carts.

The PHP function setcookie() adds headers to the HTTP response. Since headers must be inserted before the body, all setcookie() calls must be performed before any body output (usually HTML) is printed.

The following example uses a cookie to store a form value until your browser is terminated.

```
<?php
if (isset($myname))
{
    setcookie("myname", $myname);
}
print <<<_A_
```

```
<html>
....
_A_;
....
?>
```

The following example displays all the cookies releated with the visited site:

```
<?php
if (!$_COOKIE)  {  $_COOKIE = $HTTP_COOKIE_VARS;  }
foreach ($_COOKIE as $ck => $val)
{  print "<dd>Cookie name: $ck; value: $val<br />\n";  }
?>
```

Examples:

- Webmail: store and use username.
- Weather Site: store the selected city in a cookie, and next time the site is visited, the weather of the stored city is automatically displayed.

Built-in Variables

PHP has a number of built-in variables that give you access to your Web server's CGI environment, form/cookie data and PHP internals. Here are some of the most useful variables:

PHP internal variables

The $GLOBALS and $PHP_SELF variables shown in the table below are specific to PHP:

Variable Name	Description
$GLOBALS	An associative array of all global variables. This is the only variable in PHP that is available regardless of scope. You can access it anywhere without declaring it as a global first.
$PHP_SELF	This is the current script, for example /~ssb/phpinfo.php3.
$_POST	An associative array of POST variables.
$_GET	An associative array of GET variables.
$_COOKIE	An associative array of COOKIE variables.
$_SERVER	An associative array of SERVER variables.

CGI / Web server provided variables

The variables listed below are derived from CGI protocols.

Variable Name	Description
$DOCUMENT_ROOT	Your Web server's base directory with user-visible files.
$REQUEST_METHOD	The HTTP method used to access this page, for example GET or POST.
$REQUEST_URI	Full local part of the request URL, including parameters.
$SCRIPT_FILENAME	File name of the top-level page being executed.
$SCRIPT_NAME	Local URI part of the page being executed.
$SERVER_NAME	Domain name for the server.
$SERVER_PORT	TCP port number the server runs on.

HTTP Request Variables

HTTP Request Variables are derived from their corresponding HTTP headers.

$REMOTE_ADDR	IP address of the client (browser) machine
$HTTP_HOST	Host name in the browser's "location" field.
$HTTP_USER_AGENT	User agent (browser) being used.
$HTTP_REFERER	URL of the referring page.

Regular Expressions

Regular expressions, a powerful tool for manipulating text and data, are found in scripting languages, editors, programming environments, and specialized tools.

Special characters: "/" "-" "^" "." "\" "^" "$" "*" "+" "(" ")"

	Equivalent	Description or Meaning	Examples
/abc/		search for substring "abc"	
/[akt]/		search for one of "a", "k" or "t".	
/[^akt]/		true if string do not have any one of "a", "k" or "t".	
/[0-4]/	/[01234]/	search for one of "0", "1", "2", "3" or "4"	
/.../i		Case insensitive	
*		Zero or more	/abck*pqr/
+		One or more	/abck+pqr/
\d	[0-9]	Any decimal digit [0123456789]	
\D		Any character that is not a decimal digit	
\w	[0-9A-Za-z_]	Alphanumerics and underscore	
\W		Any "non-word" character	
\s	[\r\t\n\f]	Any Whitespace character	
\S		Any character that is not a whitespace character	
\b		At word boundary (at the beginning or end)	/\babc/ or /abc\b/ or /\babc\b/
\r		Return (CR)	
\f		Linefeed (LF)	
\n		New line (CRLF)	
\t		Tab	
/^.../		Starts with	/^klm/
/...$/		Ends with	/ali$/ /^good$/i
/\d{4}/		4 digits /\d\d\d\d/	/[A-Z]{3}\d{3}/
/\d{2,4}/		2 to 4 digits	/\d{2} [A-Z]{1,3} \d{2,4}/
(...)		Grouping	

Regular Expression Functions in PHP

preg_match: Perform a regular expression match

Examples:

```
// The "i" after the pattern delimiter indicates a case-insensitive search
if (preg_match ("/php/i", "PHP is the web scripting language of choice."))
{  print "A match was found.";  }
else
{  print "A match was not found.";  }

// The \b in the pattern indicates a word boundary, so only the distinct
// word "web" is matched, and not a word partial like "webbing" or "cobweb"
if (preg_match ("/\bweb\b/i", "PHP is the web scripting language of choice."))
{  print "A match was found.";  }
else
{  print "A match was not found.";  }

if (preg_match ("/\bweb\b/i", "PHP is the website scripting language of
choice."))
{  print "A match was found.";  }
else
{  print "A match was not found.";  }
```

preg_replace: Perform a regular expression search and replace

Examples:

```
$string = 'April 15, 2003';
$pattern = '/(\w+) (\d+), (\d+)/i';
$replacement = '${1}1,$3';
print  preg_replace($pattern, $replacement, $string);  // April1,2003
```

Database Handling

Communication with MySQL

PHP and MySQL are often referred to as the "dynamic duo" of dynamic Web scripting. PHP and MySQL work very well together, in addition to the speed and features of each individual tool.

The steps in using MySQL databases in PHP are as follows:

1. Connect to MySQL Server and select database (usually placed in main of the PHP scripts):
```
$dblink = mysql_connect("localhost", "db_user", "my_password")
     or die("Could not connect");
mysql_select_db("my_db")  or die("Can not select database");
```
2. Prepare and submit a query:
```
$query = "select * from person where p_city = '$city'
             order by p_lname";
$qno = mysql_query($query)
     or die("Query error: <b>$query</b><hr />" . mysql_error());
```
3. Get number of records selected / affected:
```
$n = mysql_num_rows ($qno);   // Get number of records affected.
```
4. Fetch data from database server:
```
while ($record = mysql_fetch_assoc($qno))
{
  extract($record);    // Create variables by table column names
  ........            // Use these created variables
}
```
5. Free the query resources:
```
mysql_free_result ($qno);      // Free query result set.
```
6. Close connection to database server:
```
mysql_close($dblink);          // Close connection.
```

MySQL Example

The following is a simple example of how to dump the contents of a MySQL table using PHP. The example assumes you have a MySQL user called "db_user" who can connect with password "my_password" from localhost. In the example below, PHP implements the following procedure:

```
<?php
  // Connect and select database:
  $dblink = mysql_connect("localhost", "db_user", "my_password")
       or die("Could not connect");
  mysql_select_db("my_db")  or die("Can not select database");

  // Perform SQL query:
  $query = "select * from user";
  $qno = mysql_query($query)
       or die("Query failed: $query<hr />" . mysql_error();

  $n = mysql_num_rows ($qno);       // Get number of records selected/affected.

  // Output results in HTML:
  print "<table border='1'>\n";
  while ($user = mysql_fetch_assoc($qno))    // While a record is retrieved...
  {
    // Print column headings if first row
    if (!$header_printed)
    {
      print "\t<tr>\n";
      $fields = array_keys ($user);
```

```
        reset($user);       // Rewind for the next foreach
        foreach ($fields as $field)
        {  print " <th>$field</th>\n";  }
        print " </tr>\n";
        $header_printed = true;
      }  // end of print header.
      // Output data in the record retrieved.
      print "\t<tr>\n";
      foreach ($user as $col_value)
      {
        print "\t\t<td>$col_value</td>\n";
      }
      print "\t</tr>\n";
    }
    print "</table>\n";

    mysql_free_result($qno);          // Free resultset.
    mysql_close($dblink);             // Close connection.
  ?>
```

Communication with Other Databases

Unlike other scripting languages for Web page development, PHP is open-source, cross-platform, and offers excellent connectivity to most of today's common databases including Oracle, Sybase, MySQL, ODBC (and others). PHP also offers integration with various external libraries which enable the developer to do anything from generating PDF documents to parsing XML.