



## **CENTRE FOR ADVANCED COMPUTER LEARNING AND DEVELOPMENT**

### **WEB DEVELOPMENT COURSE** **PHP STUDY MATERIAL**

#### **PHP — INTRODUCTION**

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

#### **Common Uses of PHP**

PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them. The other uses of PHP are:

- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.
- You add, delete, modify elements within your database thru PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

## **Characteristics of PHP**

Five important characteristics make PHP's practical nature possible:

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

## **"Hello World" Script in PHP**

To get a feel of PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this:

```
<html>

<head>
<title>Hello World</title>

<body>

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

```
</body>
```

```
</html>
```

It will produce the following result:

Hello, World!

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

All PHP code must be included inside one of the three special markup tags are recognized by the PHP Parser.

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

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

```
<script language="php"> PHP code goes here </script>
```

Most common tag is the `<?php...?>` and we will also use the same tag in our tutorial.

From the next chapter, we will start with PHP Environment Setup on your machine and then we will dig out almost all concepts related to PHP to make you comfortable with the PHP language.

## PHP — ENVIRONMENT SETUP

In order to develop and run PHP Web pages, three vital components need to be installed on your computer system.

**Web Server** - PHP will work with virtually all **Web Server** software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server. Download Apache for free here: <http://httpd.apache.org/download.cgi>

**Database** - PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database. Download MySQL for free here: <http://www.mysql.com/downloads/index.html>

**PHP Parser** - In order to process PHP script instructions, generate HTML output that can be sent to the **Web Browser**. to install PHP parser on your computer.

a parser must be installed to This tutorial will guide you how

## PHP Parser Installation

Before you proceed, it is important to make sure that you have a proper environment setup on your machine to develop your web programs using PHP.

Type the following address into your browser's address box.

`http://127.0.0.1/info.php`

If this displays a page showing your PHP installation related information, then it means you have PHP and Webserver installed properly. Otherwise you have to follow given procedure to install PHP on your computer.

This section will guide you to install and configure PHP over the following four platforms:

- [PHP Installation on Linux or Unix with Apache](#)
- [PHP Installation on Mac OS X with Apache](#)
- [PHP Installation on Windows NT/2000/XP with IIS](#)
- [PHP Installation on Windows NT/2000/XP with Apache](#)

## PHP Installation on Linux or Unix with Apache

If you plan to install PHP on Linux or any other variant of Unix, then here is the list of prerequisites:

- The PHP source distribution <http://www.php.net/downloads.php> The latest Apache source distribution <http://httpd.apache.org/download.cgi>
- A working PHP-supported database, if you plan to use one ( For example MySQL, Oracle etc. )
- Any other supported software to which PHP must connect (mail server,

BCMath package, JDK, and so forth)

- An ANSI C compiler
- Gnu make utility - you can freely download it at <http://www.gnu.org/software/make>

Now here are the steps to install Apache and PHP5 on your Linux or Unix machine. If your PHP or Apache versions are different, then please take care accordingly.

- If you haven't already done so, unzip and untar your Apache source distribution. Unless you have a reason to do otherwise, /usr/local is the standard place.

```
gunzip -c apache_1.3.x.tar.gz
tar -xvf apache_1.3.x.tar
```

- Build the apache Server as follows

```
cd apache_1.3.x

./configure --prefix=/usr/local/apache --enable-so make
make install
```

- Unzip and untar your PHP source distribution. Unless you have a reason to do otherwise, /usr/local is the standard place.

```
gunzip -c php-5.x.tar.gz
tar -xvf php-5.x.tar
```

```
cd php-5.x
```

- Configure and Build your PHP, assuming you are using MySQL database.

```
./configure --with-apxs=/usr/sbin/apxs \

--with-mysql=/usr/bin/mysql
make
make install
```

- Install the php.ini file. Edit this file to get configuration directives:

```
cd ../../php-5.x
cp php.ini-dist /usr/local/lib/php.ini
```

- Tell your Apache server where you want to serve files from, and what extension(s) you want to identify PHP files. **.php** is the standard, but you can use .html, .phtml, or whatever you want.
- Go to your HTTP configuration files (/usr/local/apache/conf or whatever your path is)
- Open httpd.conf with a text editor.

Search for the word DocumentRoot (which should appear twice), and change both paths to the directory you want to serve files out of (in our case, /home/httpd). We recommend a home directory rather than the default /usr/local/apache/htdocs because it is more secure, but it doesn't have to be in a home directory. You will keep all your PHP files in this directory.

- Add at least one PHP extension directive, as shown in the first line of code that follows. In the second line, we've also added a second handler to have all HTML files parsed as PHP

```
AddType application/x-httpd-php .php
```

```
AddType application/x-httpd-php .html
```

- Restart your server. Every time you change your HTTP configuration or php.ini files, you must stop and start your server again.

- 

```
cd ../bin
```

```
./apachectl start
```

- Set the document root directory permissions to world-executable. The actual PHP files in the directory need only be world-readable (644). If necessary, replace /home/httpd with your document root below:

```
chmod 755 /home/httpd/html/php
```

- Open a text editor. Type: <?php phpinfo(); ?>. Save this file in your Web server's document root as info.php.

- Start any Web browser and browse the file. You must always use an HTTP request

(http://www.testdomain.com/info.php or http://localhost/info.php or http://127.0.0.1/info.php) rather than a filename (/home/httpd/info.php) for the file

to be parsed correctly

You should see a long table of information about your new PHP installation message Congratulations!

## PHP Installation on Mac OS X with Apache

Mac users have the choice of either a binary or a source installation. In fact, your OS X probably came with Apache and PHP preinstalled. This is likely to be quite an old build, and it probably lacks many of the less common extensions.

However, if all you want is a quick Apache + PHP + MySQL/PostgreSQL setup on your laptop, this is certainly the easiest way to fly. All you need to do is edit your Apache configuration file and turn on the Web server.

So just follow the steps given below:

- Open the Apache config file in a text editor as root. `sudo open -a TextEdit /etc/httpd/httpd.conf`
- Edit the file. Uncomment the following lines:  
  
`LoadModule php5_module`  
  
`AddModule mod_php5.c`  
  
`AddType application/x-httpd-php .php`
- You may also want to uncomment the `<Directory /home/*/Sites>` block or otherwise tell Apache which directory to serve out of.
- Restart the Web server
  - `sudo apachectl graceful`
- Open a text editor. Type: `<?php phpinfo(); ?>`. Save this file in your Web server's

document root as `info.php`.

- Start any Web browser and browse the file. you must always use an HTTP request

(`http://www.testdomain.com/info.php` or `http://localhost/info.php` or `http://127.0.0.1/info.php`) rather than a filename (`/home/httpd/info.php`) for the file

to be parsed correctly

You should see a long table of information about your new PHP installation message Congratulations!

### PHP Installation on Windows NT/2000/XP with IIS

The Windows server installation of PHP running IIS is much simpler than on Unix, since it involves a precompiled binary rather than a source build.

If you plan to install PHP over Windows, then here is the list of prerequisites:

- A working PHP-supported Web server. Under previous versions of PHP, IIS/PWS was the easiest choice because a module version of PHP was available for it; but PHP now has added a much wider selection of modules for Windows.
- A correctly installed PHP-supported database like MySQL or Oracle etc. (if you plan to use one)
- The PHP Windows binary distribution (download it [atwww.php.net/downloads.php](http://www.php.net/downloads.php))
- A utility to unzip files (search <http://download.cnet.com> for PC file compression utilities)

Now here are the steps to install Apache and PHP5 on your Windows machine. If your PHP version is different, then please take care accordingly.

- Extract the binary archive using your unzip utility; C:\PHP is a common location.
- Copy some .dll files from your PHP directory to your systems directory (usually C:\Winnt\System32). You need php5ts.dll for every case. You will also probably need to copy the file corresponding to your Web server module - C:\PHP\Sapi\php5isapi.dll. It's possible you will also need others from the dlls subfolder - but start with the two mentioned above and add more if you need them.
- Copy either php.ini-dist or php.ini-recommended (preferably the latter) to your Windows directory (C:\Winnt or C:\Winnt40), and rename it php.ini. Open this file in a text editor (for example, Notepad). Edit this file to get configuration directives; We highly recommend new users set error reporting to E\_ALL on their development machines at this point. For now, the most important thing is the doc\_root directive under



the Paths and Directories section. make sure this matches your IIS Inetpub folder (or wherever you plan to serve out of).

- Stop and restart the WWW service. Go to the Start menu -> Settings -> Control Panel -> Services. Scroll down the list to IIS Admin Service. Select it and click Stop. After it stops, select World Wide Web Publishing Service and click Start. Stopping and restarting the service from within Internet Service Manager will not suffice. Since this is Windows, you may also wish to reboot.
- Open a text editor. Type: `<?php phpinfo(); ?>`. Save this file in your Web server's document root as info.php.
- Start any Web browser and browse the file. you must always use an HTTP request

(<http://www.testdomain.com/info.php> or <http://localhost/info.php> or <http://127.0.0.1/info.php>) rather than a filename (`/home/httpd/info.php`) for the file to be parsed correctly

You should see a long table of information about your new PHP installation message Congratulations!

### PHP Installation on Windows NT/2000/XP with Apache

To install Apache with PHP 5 on Windows follow the following steps. If your PHP and Apache versions are different, then please take care accordingly.

- Download Apache server from [www.apache.org/dist/httpd/binaries/win32](http://www.apache.org/dist/httpd/binaries/win32). You want the current stable release version with the no\_src.msi extension. Double-click the installer file to install; C:\Program Files is a common location. The installer will also ask you whether you want to run Apache as a service or from the command line or DOS prompt. We recommend you do not install as a service, as this may cause problems with startup.
- Extract the PHP binary archive using your unzip utility; C:\PHP is a common location.
- Copy some .dll files from your PHP directory to your system directory (usually C:\Windows). You need php5ts.dll for every case. You will also probably need to copy the file corresponding to your Web server module - C:\PHP\Sapi\php5apache.dll. to your Apache modules directory. It's possible that you will also need others from the dlls subfolder, but start with the two mentioned previously and add more if you need them.

- Copy either php.ini-dist or php.ini-recommended (preferably the latter) to your Windows directory, and rename it php.ini. Open this file in a text editor (for example, Notepad). Edit this file to get configuration directives; At this point, we highly recommend that new users set error reporting to E\_ALL on their development machines.
- Tell your Apache server where you want to serve files from and what extension(s) you want to identify PHP files (.php is the standard, but you can use .html, .phtml, or whatever you want). Go to your HTTP configuration files (C:\Program Files\Apache Group\Apache\conf or whatever your path is), and open httpd.conf with a text editor. Search for the word DocumentRoot (which should appear twice) and change both paths to the directory you want to serve files out of. (The default is C:\Program Files\Apache Group\Apache\htdocs.). Add at least one PHP extension directive as shown in the first line of the following code:

```
LoadModule php5_module modules/php5apache.dll AddType
application/x-httpd-php .php .phtml
```

- You may also need to add the following line:

```
AddModule mod_php5.c
```

- Stop and restart the WWW service. Go to the Start menu -> Settings -> Control Panel -> Services. Scroll down the list to IIS Admin Service. Select it and click Stop. After it stops, select World Wide Web Publishing Service and click Start. Stopping and restarting the service from within Internet Service Manager will not suffice. Since this is Windows, you may also wish to reboot.
- Open a text editor. Type: <?php phpinfo(); ?>. Save this file in your Web server's document root as info.php.
- Start any Web browser and browse the file. you must always use an HTTP request

(http://www.testdomain.com/info.php or http://localhost/info.php or http://127.0.0.1/info.php) rather than a filename (/home/httpd/info.php) for the file to be parsed correctly

You should see a long table of information about your new PHP installation message Congratulations!

## **Apache Configuration**

If you are using Apache as a Web Server, then this section will guide you to edit Apache Configuration Files.

## **PHP.INI File Configuration**

The PHP configuration file, php.ini, is the final and most immediate way to affect PHP's functionality.

## **Windows IIS Configuration**

To configure IIS on your Windows machine you can refer your IIS Reference Manual shipped along with IIS.

## **Apache Configuration for PHP**

Apache uses httpd.conf file for global settings, and the .htaccess file for per-directory access settings. Older versions of Apache split up httpd.conf into three files (access.conf, httpd.conf, and srm.conf), and some users still prefer this arrangement.

Apache server has a very powerful, but slightly complex, configuration system of its own. Learn more about it at the Apache Web site: [www.apache.org](http://www.apache.org)

The following section describes settings in httpd.conf that affect PHP directly and cannot be set elsewhere. If you have standard installation, then httpd.conf will be found at /etc/httpd/conf:

### **Timeout**

This value sets the default number of seconds before any HTTP request will time out. If you set PHP's max\_execution\_time to longer than this value, PHP will keep grinding away but the

user may see a 404 error. In safe mode, this value will be ignored; you must use the timeout value in php.ini instead

### **DocumentRoot**

DocumentRoot designates the root directory for all HTTP processes on that server. It looks something like this on Unix:

```
DocumentRoot ./usr/local/apache_1.3.6/htdocs.
```

You can choose any directory as document root.

## **AddType**

The PHP MIME type needs to be set here for PHP files to be parsed. Remember that you can associate any file extension with PHP like .php3, .php5 or .htm.

```
AddType application/x-httpd-php .php
```

```
AddType application/x-httpd-phps .phps
```

```
AddType application/x-httpd-php3 .php3 .phtml
```

```
AddType application/x-httpd-php .html
```

## **Action**

You must uncomment this line for the Windows apxs module version of Apache with shared object support:

```
LoadModule php4_module modules/php4apache.dll
```

or on Unix flavors:

```
LoadModule php4_module modules/mod_php.so
```

## **AddModule**

You must uncomment this line for the static module version of Apache.

```
AddModule mod_php4.c
```

## PHP.INI file Configuration

The PHP configuration file, `php.ini`, is the final and most immediate way to affect PHP's functionality. The `php.ini` file is read each time PHP is initialized. In other words, whenever `httpd` is restarted for the module version or with each script execution for the CGI version. If

your change isn't showing up, remember to stop and restart `httpd`. If it still isn't showing up, use `phpinfo()` to check the path to `php.ini`.

The configuration file is well commented and thorough. Keys are case sensitive, keyword values are not; whitespace, and lines beginning with semicolons are ignored. Booleans can be represented by 1/0, Yes/No, On/Off, or True/False. The default values in `php.ini-dist` will result in a reasonable PHP installation that can be tweaked later.

Here we are explaining the important settings in `php.ini` which you may need for your PHP Parser.

### **`short_open_tag = Off`**

Short open tags look like this: `<? ?>`. This option must be set to Off if you want to use XML functions.

### **`safe_mode = Off`**

If this is set to On, you probably compiled PHP with the `--enable-safe-mode` flag. Safe mode is most relevant to CGI use. See the explanation in the section "CGI compile-time options". earlier in this chapter.

### **`safe_mode_exec_dir = [DIR]`**

This option is relevant only if safe mode is on; it can also be set with the `--with-exec-dir` flag during the Unix build process. PHP in safe mode only executes external binaries out of this directory. The default is `/usr/local/bin`. This has nothing to do with serving up a normal PHP/HTML Web page.

### **`safe_mode_allowed_env_vars = [PHP_]`**

This option sets which environment variables users can change in safe mode. The default is only those variables prepended with `"PHP_"`. If this directive is empty, most variables are alterable.

**safe\_mode\_protected\_env\_vars = [LD\_LIBRARY\_PATH]**

This option sets which environment variables users can't change in safe mode, even if safe\_mode\_allowed\_env\_vars is set permissively.

**disable\_functions = [function1, function2...]**

A welcome addition to PHP4 configuration and one perpetuated in PHP5 is the ability to disable selected functions for security reasons. Previously, this necessitated hand-editing the C code from which PHP was made. Filesystem, system, and network functions should probably be the first to go because allowing the capability to write files and alter the system over HTTP is never such a safe idea.

**max\_execution\_time = 30**

The function set\_time\_limit() won't work in safe mode, so this is the main way to make a script time out in safe mode. In Windows, you have to abort based on maximum memory consumed rather than time. You can also use the Apache timeout setting to timeout if you use Apache, but that will apply to non-PHP files on the site too.

**error\_reporting = E\_ALL & ~E\_NOTICE**

The default value is E\_ALL & ~E\_NOTICE, all errors except notices. Development servers should be set to at least the default; only production servers should even consider a lesser value

**error\_prepend\_string = [""]**

With its bookend, error\_append\_string, this setting allows you to make error messages a different color than other text, or what you have.

**warn\_plus\_overloading = Off**

This setting issues a warning if the + operator is used with strings, as in a form value.

**variables\_order = EGPCS**

This configuration setting supersedes gpc\_order. Both are now deprecated along with register\_globals. It sets the order of the different variables: Environment, GET, POST, COOKIE, and SERVER (aka Built-in).

You can change this order around. Variables will be overwritten successively in

left-to-right order, with the rightmost one winning the hand every time. This means if you left the default setting and happened to use the same name for an environment variable, a POST variable, and a COOKIE variable, the COOKIE variable would own that name at the end of the process. In real life, this doesn't happen much.

### **register\_globals = Off**

This setting allows you to decide whether you wish to register EGPCS variables as global. This is now deprecated, and as of PHP4.2, this flag is set to Off by default. Use superglobal arrays instead. All the major code listings in this book use superglobal arrays.

### **gpc\_order = GPC**

This setting has been GPC Deprecated.

### **magic\_quotes\_gpc = On**

This setting escapes quotes in incoming GET/POST/COOKIE data. If you use a lot of forms which possibly submit to themselves or other forms and display form values, you may need to set this directive to On or prepare to use addslashes() on string-type data.

### **magic\_quotes\_runtime = Off**

This setting escapes quotes in incoming database and text strings. Remember that SQL adds slashes to single quotes and apostrophes when storing strings and does not strip them off when returning them. If this setting is Off, you will need to use stripslashes() when outputting any type of string data from a SQL database. If magic\_quotes\_sybase is set to On, this must be Off.

### **magic\_quotes\_sybase = Off**

This setting escapes single quotes in incoming database and text strings with Sybase-style single quotes rather than backslashes. If magic\_quotes\_runtime is set to On, this must be Off.

### **auto-prepend-file = [path/to/file]**

If a path is specified here, PHP must automatically include() it at the beginning of every PHP file. Include path restrictions do apply.

### **auto-append-file = [path/to/file]**

If a path is specified here, PHP must automatically include() it at the end of

every PHP file.unless you escape by using the `exit()` function. Include path restrictions do apply.

### **`include_path = [DIR]`**

If you set this value, you will only be allowed to include or require files from these directories. The include directory is generally under your document root; this is mandatory if you're running in safe mode. Set this to `.` in order to include files from the same directory your script is in. Multiple directories are separated by colons: `./usr/local/apache/htdocs:/usr/local/lib`.

### **`doc_root = [DIR]`**

If you're using Apache, you've already set a document root for this server or virtual host in `httpd.conf`. Set this value here if you're using safe mode or if you want to enable PHP only on a portion of your site (for example, only in one subdirectory of your Web root).

### **`file_uploads = [on/off]`**

Turn on this flag if you will upload files using PHP script.

### **`upload_tmp_dir = [DIR]`**

Do not uncomment this line unless you understand the implications of HTTP uploads!

### **`session.save-handler = files`**

Except in rare circumstances, you will not want to change this setting. So don't touch it.

### **`ignore_user_abort = [On/Off]`**

This setting controls what happens if a site visitor clicks the browser's Stop button. The default is On, which means that the script continues to run to completion or timeout. If the setting is changed to Off, the script will abort. This setting only works in module mode, not CGI.

### **`mysql.default_host = hostname`**

The default server host to use when connecting to the database server if no other host is specified.

### **`mysql.default_user = username`**



The default user name to use when connecting to the database server if no other name is specified.

**mysql.default\_password = password**

The default password to use when connecting to the database server if no other password is specified.

## PHP — SYNTAX OVERVIEW

### Escaping to PHP

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP.' There are four ways to do this:

#### Canonical PHP tags

The most universally effective PHP tag style is:

```
<?php...?>
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

#### Short-open (SGML-style) tags

Short or short-open tags look like this:

```
<?...?>
```

Short tags are, as one might expect, the shortest **option You must** do one of two things to enable PHP to recognize the tags:

- Choose the `--enable-short-tags` configuration option when you're building PHP.
- Set the `short_open_tag` setting in your `php.ini` file to `on`. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

#### ASP-style tags

ASP-style tags mimic the tags used by Active Server Pages to delineate code blocks. ASP-style tags look like this:

```
<%...%>
```

To use ASP-style tags, you will need to set the configuration option in your php.ini file.

## HTML script tags

HTML script tags look like this:

```
<script language="PHP">...</script>
```

## Commenting PHP Code

A *comment* is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

**Single-line comments:** They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

```
<?
#    This is a comment, and

#    This is the second line of the comment

//    This is a comment too. Each style comments only print "An example
with single line comments";

?>
```

**Multi-lines printing:** Here are the examples to print multiple lines in a single print statement:

```
<?

#    First Example print <<<END

This uses the "here document" syntax to output multiple lines with
$variable interpolation. Note that the here document terminator must
appear on a line with just a semicolon no extra whitespace! END;
#    Second Example

print "This spans
multiple lines. The newlines will be
```

```
output as well";
```

```
?>
```

**Multi-lines comments:** They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here are the example of multi lines comments.

```
<?
/* This is a comment with multiline
```

```
Author : Mohammad Mohtashim
```

```
Purpose: Multiline Comments Demo
Subject: PHP
```

```
*/
```

```
print "An example with multi line comments"; ?>
```

### PHP is whitespace insensitive

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row. one whitespace character is the same as many such characters.

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable \$four is equivalent:

\$four	=2+2;	// single spaces
\$four	<tab>=<tab>2<tab>+<tab>2 ; // spaces and tabs	
\$four	=	
2+		
2; //	multiple	lines

### PHP is case sensitive

Yeah it is true that PHP is a case sensitive language. Try out the following

example:

```
<html>

<body>

<?
$capital = 67;

print("Variable capital is $capital<br>");
print("Variable CaPiTaL is $CaPiTaL<br>");

?>

</body>
</html>
```

This will produce the following result:

Variable capital is 67

Variable CaPiTaL is

### **Statements are expressions terminated by semicolons**

A *statement* in PHP is any expression that is followed by a semicolon (;). Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called \$greeting:

```
$greeting = "Welcome to PHP!";
```

### **Expressions are combinations of tokens**

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables (\$two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth

### **Braces make blocks**

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent:

```
if (3 == 2 + 1)
```

```
print("Good - I haven't totally lost my mind.<br>");
```

```
if (3 == 2 + 1)
```

```
{
```

```
print("Good - I haven't totally");
```

```
print("lost my mind.<br>");
```

```
}
```

## Running PHP Script from Command Prompt

Yes you can run your PHP script on your command prompt. Assuming you have the following content in test.php file

```
<?php
```

```
echo "Hello PHP!!!!!";
```

```
?>
```

Now run this script as command prompt as follows:

```
$ php test.php
```

It will produce the following result

```
Hello PHP!!!!!
```

## PHP — VARIABLE TYPES

The main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign (\$).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.

- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

PHP has a total of eight data types which we use to construct our variables:

- **Integers:** are whole numbers, without a decimal point, like 4195.
- **Doubles:** are floating-point numbers, like 3.14159 or 49.1.
- **Booleans:** have only two possible values either true or false.
- **NULL:** is a special type that only has one value: NULL.
- **Strings:** are sequences of characters, like 'PHP supports string operations.'
- **Arrays:** are named and indexed collections of other values.
- **Objects:** are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources:** are special variables that hold references to resources external to PHP (such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

We will explain only simple data type in this chapters. Array and Objects will be explained separately.

## Integers

They are whole numbers, without a decimal point, like 4195. They are the simplest type .they correspond to simple whole numbers, both positive and

negative. Integers can be assigned to variables, or they can be used in expressions, like so:

```
$int_var = 12345;
```

```
$another_int = -12345 + 12345;
```

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimal have a leading 0x.

For most common platforms, the largest integer is  $(2^{31} - 1)$  (or 2,147,483,647), and the smallest (most negative) integer is  $-(2^{31} - 1)$  (or -2,147,483,647).

## Doubles

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code:

```
$many = 2.2888800;
```

```
$many_2 = 2.2111200;
```

```
$few = $many + $many_2;
```

```
print(.$many + $many_2 = $few<br>.);
```

It produces the following browser output:

```
2.28888 + 2.21112 = 4.5
```

## Boolean

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so:

```
if (TRUE)
```

```
print("This will always print<br>");
```

```
else
```

```
print("This will never print<br>");
```

## Interpreting other types as Booleans

Here are the rules for determine the "truth" of any value not already of the Boolean type:

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise.
- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.
- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;
```

```
$true_str = "Tried and true"
```

```
$true_array[49] = "An array element";
```

```
$false_array = array();
```

```
$false_null = NULL;
```

```
$false_num = 999 - 999;
```

```
$false_str = "";
```

## NULL

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this:



```
$my_var = NULL;
```

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed:

```
$my_var = null;
```

A variable that has been assigned NULL has the following properties:

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with IsSet() function.

## Strings

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string:

```
$string_1 = "This is a string in double quotes";
```

```
$string_2 = "This is a somewhat longer, singly quoted string"; $string_39 =  
"This string has thirty-nine characters"; $string_0 = ""; // a string with zero  
characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?
```

```
$variable = "name";
```

```
$literally = 'My $variable will not print!\n'; print($literally);
```

```
$literally = "My $variable will print!\n";
```

```
print($literally);
```

```
?>
```

This will produce the following result:

```
My $variable will not print!\n
```

```
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with \$) are replaced with string representations of their values.

The escape-sequence replacements are:

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \\$ is replaced by the dollar sign itself (\$)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

## Here Document

You can assign multiple lines to a single string variable using **here document**:

```
<?php

$channel =<<<_XML_

<channel>

<title>What's For Dinner<title>

<link>http://menu.example.com/<link>

<description>Choose what to eat tonight.</description>

</channel>
```

```
_XML_;
```

```
echo <<<END
```

This uses the "here document" syntax to output multiple lines with variable interpolation. Note that the here document terminator must appear on a line with just a semicolon. no extra whitespace!

```
<br />
```

```
END;
```

```
print $channel;
```

```
?>
```

This will produce the following result:

This uses the "here document" syntax to output multiple lines with variable interpolation. Note that the here document terminator must appear on a line with just a semicolon. no extra whitespace!

```
<channel>
```

```
<title>What's For Dinner<title> <link>http://menu.example.com/<link>  
<description>Choose what to eat tonight.</description>
```

## Variable Naming

Rules for naming a variable is:

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like + , - , % , ( , ) . & , etc

There is no size limit for variables.

## PHP – Variables

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

- Local variables
- Function parameters
- Global variables
- Static variables

### **PHP Local Variables**

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function:

```
<?

$x = 4;

function assignx () {

    $x = 0;

    print "\$x inside function is $x.
    ";

}

assignx();

print "\$x outside of function is $x.

";

?>
```

This will produce the following result.

\$x inside function is 0.

\$x outside of function is 4.

## PHP Function Parameters

PHP Functions are covered in detail in PHP Function Chapter. In short, a function is a small unit of program which can take some input in the form of parameters and does some processing and may return a value.

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be:

```
<?

    ○ //    multiply a value by 10 and return it to the caller function multiply
      ($value) {

$value = $value * 10; return $value;

}

$retval = multiply (10);

Print "Return value is $retval\n";

?>
```

This will produce the following result.

```
Return value is 100
```

## PHP Global Variables

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword **GLOBAL** in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name. Consider an example:

```
<?

$somevar = 15;

function addit() {
```

```

GLOBAL $somevar;

$somevar++;

print "Somevar is $somevar";

}

addit();

?>

```

This will produce the following result.

```

Somevar is 16

```

## PHP Static Variables

The final type of variable scoping that I discuss is known as static. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

You can declare a variable to be static simply by placing the keyword `STATIC` in front of the variable name.

```

<?

function keep_track() {

    STATIC $count = 0;

    $count++;

    print $count;

    print "
    ";
}

keep_track();

keep_track();

keep_track();

```

```
?>
```

This will produce the following result.

```
1
```

```
2
```

```
3
```

## PHP — CONSTANTS

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use `define()` function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a `$`. You can also use the function `constant()` to read a constant's value if you wish to obtain the constant's name dynamically.

### **constant() function**

As indicated by the name, this function will return the value of the constant.

This is useful when you want to retrieve value of a constant, but you do not know its name, i.e., it is stored in a variable or returned by a function.

### **constant() example**

```
<?php
```

```
define("MINSIZE", 50);
```

```
echo MINSIZE;
```

```
echo constant("MINSIZE"); // same thing as the previous line
```

?>

Only scalar data (boolean, integer, float and string) can be contained in constants.

### Differences between constants and variables are

- There is no need to write a dollar sign (\$) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the define() function.
- Constants may be defined and accessed anywhere without regard to variable scoping rules.
- Once the Constants have been set, may not be redefined or undefined.

### Valid and invalid constant names

```
// Valid constant names

define("ONE", "first thing");

define("TWO2", "second thing");

define("THREE_3", "third thing")

// Invalid constant names

define("2TWO", "second thing");

define("__THREE__", "third value");
```

### PHP Magic constants

PHP provides a large number of predefined constants to any script which it runs.

There are five magical constants that change depending on where they are used. For example, the value of `__LINE__` depends on the line that it's used on in your script. These special constants are case-insensitive and are as follows:

The following table lists a few "magical" PHP constants along with their description:



Name	Description
__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, __FILE__ always contains an absolute path whereas in older versions it contained relative path under some circumstances.
__FUNCTION__	The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__CLASS__	The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

## PHP — OPERATOR TYPES

**What is Operator?** Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators

- Assignment Operators
- Conditional (or ternary) Operators Let's have a look on all operators one by one.

## Arithmetic Operators

The following arithmetic operators are supported by PHP language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide the numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

## Example

Try the following example to understand all the arithmetic operators. Copy and paste following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>

<head><title>Arithmetical Operators</title><head>

<body>

<?php

$a = 42;

$b = 20;


$c = $a + $b;

echo "Addition Operation Result: $c <br/>"; $c = $a - $b;

echo "Subtraction Operation Result: $c <br/>"; $c = $a * $b;

echo "Multiplication Operation Result: $c <br/>"; $c = $a / $b;

echo "Division Operation Result: $c <br/>"; $c = $a % $b;

echo "Modulus Operation Result: $c <br/>";

$c = $a++;

echo "Increment Operation Result: $c <br/>"; $c = $a--;

echo "Decrement Operation Result: $c <br/>";

?>

</body>

</html>
```

This will produce the following result:

Addition Operation Result: 62

Subtraction Operation Result: 22

Multiplication Operation Result: 840

Division Operation Result: 2.1

Modulus Operation Result: 2

Increment Operation Result: 42

Decrement Operation Result: 43