PhP

hypertext preprocessor

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web-based software applications. This tutorial will help you understand the basics of PHP and how to put it in practice.

Audience

This tutorial has been designed to meet the requirements of all those readers who are keen to learn the basics of PHP.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of computer programming, Internet, Database, and MySQL.

Copyright & Disclaimer

Table of Contents

About the Tutorial i
Audience i
Prerequisites i
Copyright & Disclaimer i
Table of Contents ii

PART 1: LEARNING PHP 1

1. **PHP – Introduction 2**
Common Uses of PHP 2
Characteristics of PHP 3
"Hello World" Script in PHP 3

2. **PHP – Environment Setup 4**
PHP Parser Installation 4
PHP Installation on Linux or Unix with Apache 4
PHP Installation on Mac OS X with Apache 6
PHP Installation on Windows NT/2000/XP with IIS 7
PHP Installation on Windows NT/2000/XP with Apache 8
Apache Configuration for PHP 10
PHP.INI file Configuration 11

3. **PHP – Syntax Overview 15**
Escaping to PHP 15
Commenting PHP Code 16
PHP is whitespace insensitive 16
PHP is case sensitive 17
Statements are expressions terminated by semicolons 17
Expressions are combinations of tokens 17
Braces make blocks 17
Running PHP Script from Command Prompt 18

4. **PHP – Variable Types 19**
Integers 20
Doubles 20
Boolean 20
NULL 21
Strings 21
Variable Naming 24
PHP – Variables 24
PHP Local Variables 24
PHP Function Parameters 25
PHP Global Variables 25
PHP Static Variables 26

5. **PHP – Constants 27**

6. **PHP – Operator Types 29**
Arithmetic Operators 29

Comparison Operators 31
Logical Operators 33
Assignment Operators 35
Conditional Operator 37
Operators Categories 38
Precedence of PHP Operators 38

7. **PHP – Decision Making 40**
The If...Else Statement 40
The ElseIf Statement 41
The Switch Statement 42

8. **PHP – Loop Types 45**
The for loop statement 45
The while loop statement 46
The do...while loop statement 47
The foreach loop statement 48
The break statement 48
The continue statement 50

9. **PHP – Arrays 52**
Numeric Array 52
Associative Arrays 53
Multidimensional Arrays 54

10. **PHP – Strings 57**
String Concatenation Operator 58
Using the strlen() function 58
Using the strpos() function 59

11. **PHP – Web Concepts 60**
Identifying Browser & Platform 60
Display Images Randomly 61
Using HTML Forms 62
Browser Redirection 63
Displaying "File Download" Dialog Box 64

12. **PHP – GET and POST Methods 66**
The GET Method 66
The POST Method 67
The $_REQUEST variable 68

13. **PHp – File Inclusion 70**
The include() Function 70
The require() Function 71

14. **PHP – Files & I/O 72**
Opening and Closing Files 72
Reading a file 73
Writing a File 74

15. **PHP – Functions 76**
Creating PHP Function 76
PHP Functions with Parameters 77
Passing Arguments by Reference 77

PHP Functions returning value 78
Setting Default Values for Function Parameters 79
Dynamic Function Calls 80

16. PHP – Cookies 81
The Anatomy of a Cookie 81
Setting Cookies with PHP 82
Accessing Cookies with PHP 83
Deleting Cookie with PHP 84

17. PHP – Sessions 85
Starting a PHP Session 85
Destroying a PHP Session 87
Turning on Auto Session 87
Sessions without cookies 87

18. PHP – Sending Emails 89
Sending plain text email 89
Sending HTML email 91
Sending attachments with email 92

19. PHP – File Uploading 95
Creating an Upload Form 95
Creating an upload script 96

20. PHP – Coding Standard 98

PART 2: ADVANCED PHP 101

21. PHP – Predefined Variables 102
PHP Superglobals 102
Server variables: $_SERVER 103

22. PHP – Regular Expression 106
POSIX Regular Expressions 106
PHP's Regexp POSIX Functions 108
PHP – Function ereg() 108
PHP – Function ereg_replace() 110
PHP – Function eregi() 111
PHP – Function eregi_replace() 112
PHP – Function split() 113
PHP – Function spliti() 114
PHP – Function sql_regcase() 115
PERL Style Regular Expressions 115
PHP's Regexp PERL Compatible Functions 117
PHP – Function preg_match() 117
PHP – Function preg_match_all() 118
PHP – Function preg_replace() 119
PHP – Function preg_split() 120
PHP – Function preg_grep() 121
PHP – Function preg_quote() 122

23. PHP – Error and Exception Handling 124
Using die() function 124

Defining Custom Error Handling Function 124
Exceptions Handling 127

24. PHP – Bugs Debugging 129

25. PHP – Date and Time 131
Getting the Time Stamp with time() 131
Converting a Time Stamp with getdate() 131
Converting a Time Stamp with date() 133

26. PHP – PHP and MySQL 136
What you should already have? 136
Connecting to MySQL Database 137
Create MySQL Database Using PHP 138
Delete MySQL Database Using PHP 143
Insert Data to MySQL Database 144
Retrieving Data from MySQL Database 148
Using Paging through PHP 152
Updating Data into MySQL Database 154
Deleting Data from MySQL Database 157
Using PHP to Backup MySQL Database 159

27. PHP – PHP and AJAX 162
What is AJAX ? 162
PHP and AJAX Example 162
Client Side HTML file 163
Server Side PHP file 165

28. PHP – PHP and XML 167
HTML list that's not valid XML 167
HTML list that is valid XML 167
Parsing an XML Document 167
Generating an XML Document 169

29. PHP – Object Oriented Programming 170
Object Oriented Concepts 170
Defining PHP Classes 171
Creating Objects in PHP 172
Calling Member Functions 172
Constructor Functions 173
Destructor 174
Inheritance 174
Function Overriding 175
Public Members 175
Private members 175
Protected members 176
Interfaces 177
Constants 177
Abstract Classes 177
Static Keyword 178
Final Keyword 178

30. PHP – PHP for C Developers 181
Similarities 181
Differences 181

31. **PHP – PHP for PERL Developers 183**
Similarities 183
Differences 183

PART 3: FUNCTION REFERENCE 185

32. **PHP – Function Reference 186**

Part 1: Learning PHP

1. 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 ate 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.

2. 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, a parser must be installed to generate HTML output that can be sent to the **Web Browser.** This tutorial will guide you how to install PHP parser on your computer.

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:

```
Load Module 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)

- 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. 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.

Just Check it here: PHP.INI File Configuration

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

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.

3. 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>=<tab2<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!!!!!
```

4. 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 simile 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 hexadecimals 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
```

5. 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).

6. 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
```

Comparison Operators

There are following comparison operators supported by PHP language.

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

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes, then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal, then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes, then condition becomes true.	(A <= B) is true.

Example

Try the following example to understand all the comparison operators. Copy and paste the 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>Comparison Operators</title><head>
<body>
<?php
    $a = 42;
    $b = 20;

    if( $a == $b ){
       echo "TEST1 : a is equal to b<br/>";
    }else{
       echo "TEST1 : a is not equal to b<br/>";
    }

    if( $a > $b ){
       echo "TEST2 : a is greater than  b<br/>";
    }else{
       echo "TEST2 : a is not greater than b<br/>";
    }
    if( $a < $b ){
       echo "TEST3 : a is less than  b<br/>";
    }else{
       echo "TEST3 : a is not less than b<br/>";
    }
    if( $a != $b ){
       echo "TEST4 : a is not equal to b<br/>";
    }else{
       echo "TEST4 : a is equal to b<br/>";
    }
    if( $a >= $b ){
       echo "TEST5 : a is either greater than or equal to b<br/>";
    }else{
       echo "TEST5 : a is neither greater than nor equal to b<br/>";
    }
    if( $a <= $b ){
       echo "TEST6 : a is either less than or equal to b<br/>";
    }else{
       echo "TEST6 : a is neither less than nor equal to b<br/>";
    }
?>
</body>
</html>
```

This will produce the following result:

```
TEST1 : a is not equal to b
TEST2 : a is greater than b
TEST3 : a is not less than b
TEST4 : a is not equal to b
TEST5 : a is either greater than or equal to b
```

```
TEST6 : a is neither less than nor equal to b
```

Logical Operators

The following logical operators are supported by PHP language.

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

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true, then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero, then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero, then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is false.

Example

Try the following example to understand all the logical operators. Copy and paste the 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>Logical Operators</title><head>
<body>
<?php
    $a = 42;
    $b = 0;

    if( $a && $b ){
       echo "TEST1 : Both a and b are true<br/>";
    }else{
       echo "TEST1 : Either a or b is false<br/>";
    }
    if( $a and $b ){
       echo "TEST2 : Both a and b are true<br/>";
    }else{
       echo "TEST2 : Either a or b is false<br/>";
    }
    if( $a || $b ){
       echo "TEST3 : Either a or b is true<br/>";
    }else{
       echo "TEST3 : Both a and b are false<br/>";
    }
    if( $a or $b ){
       echo "TEST4 : Either a or b is true<br/>";
    }else{
       echo "TEST4 : Both a and b are false<br/>";
    }
    $a = 10;
    $b = 20;
    if( $a ){
       echo "TEST5 : a is true <br/>";
    }else{
       echo "TEST5 : a  is false<br/>";
    }
    if( $b ){
       echo "TEST6 : b is true <br/>";
    }else{
       echo "TEST6 : b  is false<br/>";
    }
    if( !$a ){
       echo "TEST7 : a is true <br/>";
    }else{
       echo "TEST7 : a  is false<br/>";
    }
    if( !$b ){
       echo "TEST8 : b is true <br/>";
    }else{
       echo "TEST8 : b  is false<br/>";
    }
?>
</body>
</html>
```

This will produce the following result:

```
TEST1 : Either a or b is false
TEST2 : Either a or b is false
TEST3 : Either a or b is true
TEST4 : Either a or b is true
TEST5 : a is true
TEST6 : b is true
TEST7 : a is false
TEST8 : b is false
```

Assignment Operators

PHP supports the following assignment operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A

*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Example

Try the following example to understand all the assignment operators. Copy and paste the 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>Assignment Operators</title><head>
<body>
<?php
    $a = 42;
    $b = 20;

    $c = $a + $b;   /* Assignment operator */
    echo "Addition Operation Result: $c <br/>";
    $c += $a;  /* c value was 42 + 20 = 62 */
    echo "Add AND Assignment Operation Result: $c <br/>";
    $c -= $a; /* c value was 42 + 20 + 42 = 104 */
    echo "Subtract AND Assignment Operation Result: $c <br/>";
    $c *= $a; /* c value was 104 - 42 = 62 */
    echo "Multiply AND Assignment Operation Result: $c <br/>";
    $c /= $a;  /* c value was 62 * 42 = 2604 */
    echo "Division AND Assignment Operation Result: $c <br/>";
    $c %= $a; /* c value was 2604/42 = 62*/
    echo "Modulus AND Assignment Operation Result: $c <br/>";
?>
</body>
</html>
```

This will produce the following result:

```
Addition Operation Result: 62
Add AND Assignment Operation Result: 104
Subtract AND Assignment Operation Result: 62
Multiply AND Assignment Operation Result: 2604
Division AND Assignment Operation Result: 62
Modulus AND Assignment Operation Result: 20
```

Conditional Operator

There is one more operator called the conditional operator. It first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Try the following example to understand the conditional operator. Copy and paste the 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 = 10;
    $b = 20;

    /* If condition is true then assign a to result otherwise b */
    $result = ($a > $b ) ? $a :$b;
    echo "TEST1 : Value of result is $result<br/>";
    /* If condition is true then assign a to result otherwise b */
    $result = ($a < $b ) ? $a :$b;
    echo "TEST2 : Value of result is $result<br/>";
?>
</body>
</html>
```

This will produce the following result:

```
TEST1 : Value of result is 20
TEST2 : Value of result is 10
```

Operators Categories

All the operators we have discussed above can be categorized into the following categories:

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

Precedence of PHP Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator: For example, x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	**Operator**	**Associativity**
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right

Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

7. PHP – Decision Making

The if, elseif ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports the following three decision making statements:

- **if...else statement** - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **elseif statement** - is used with the if...else statement to execute a set of code if **one** of several condition are true
- **switch statement** - is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

Syntax

```
if (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

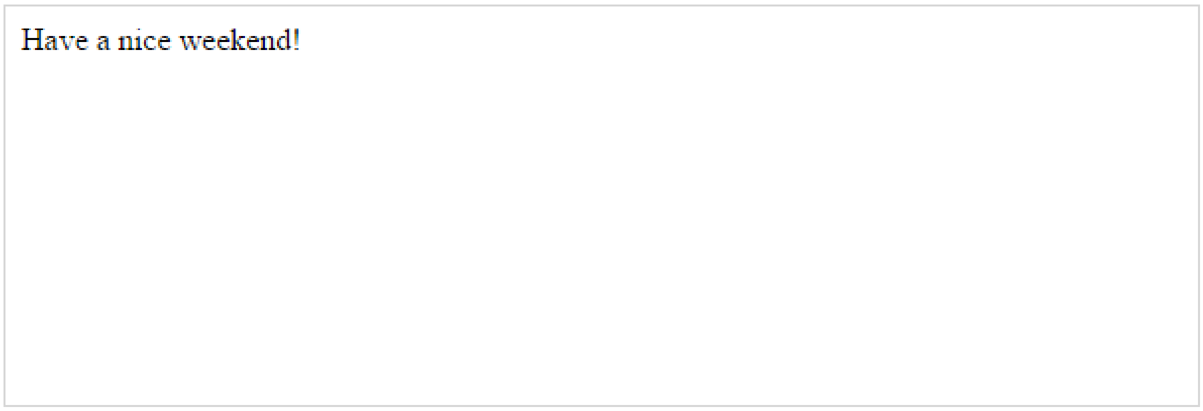
Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

```
<html>
<body>
    <?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
else
  echo "Have a nice day!";
?>
```

```
    </body>


</html>
```

It will produce the following result:



```
Have a nice weekend!
```

The ElseIf Statement

If you want to execute some code if one of the several conditions is true, then use the elseif statement.

Syntax

```
if (condition)
  code to be executed if condition is true;
elseif (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

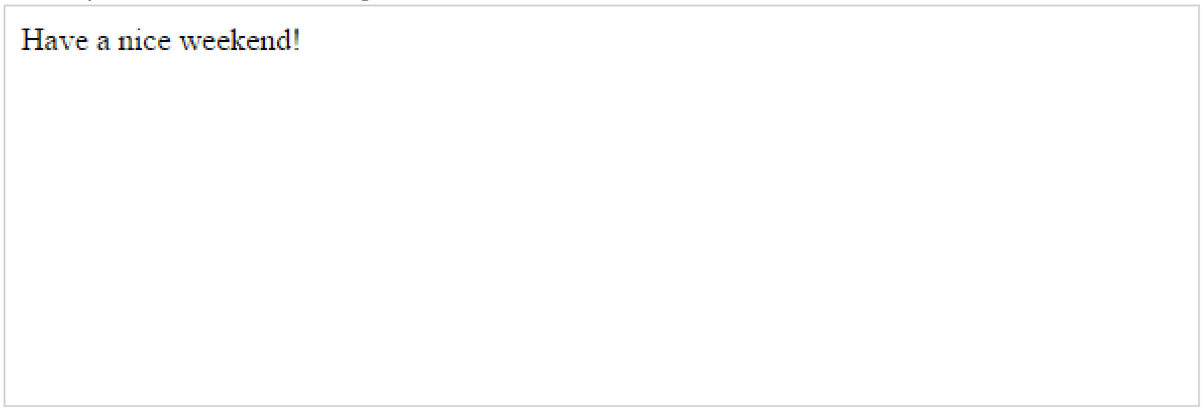
Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>
    <?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
```

```
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>
    </body>
</html>
```

It will produce the following result:



```
Have a nice weekend!
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

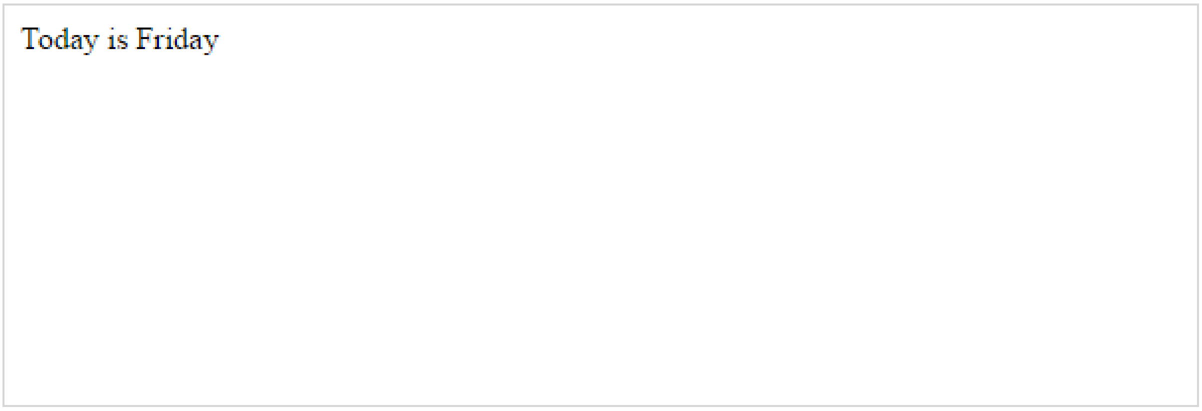
```
switch (expression)
{
case label1:
  code to be executed if expression = label1;
  break;
case label2:
  code to be executed if expression = label2;
  break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
```

Example

The *switch* statement works in an unusual way. First it evaluates the given expression, then seeks a label to match the resulting value. If a matching value is found, then the code associated with the matching label will be executed. If none of the labels match, then the statement will execute any specified default code.

```
<html>
<body>
    <?php
$d=date("D");
switch ($d)
{
case "Mon":
  echo "Today is Monday";
  break;
case "Tue":
  echo "Today is Tuesday";
  break;
case "Wed":
  echo "Today is Wednesday";
  break;
case "Thu":
  echo "Today is Thursday";
  break;
case "Fri":
  echo "Today is Friday";
  break;
case "Sat":
  echo "Today is Saturday";
  break;
case "Sun":
  echo "Today is Sunday";
  break;
default:
  echo "Wonder which day is this ?";
}
?>
    </body>
</html>
```

It will produce the following result:



```
Today is Friday
```

8. PHP – Loop Types

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for -** loops through a block of code a specified number of times.
- **while -** loops through a block of code if and as long as a specified condition is true.
- **do...while -** loops through a block of code once, and then repeats the loop as long as a special condition is true.
- **foreach -** loops through a block of code for each element in an array.

We will discuss about **continue** and **break** keywords used to control the loops execution.

The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

Syntax

```
for (initialization; condition; increment)
{
  code to be executed;
}
```

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop:

```
<html>
<body>
<?php
$a = 0;
$b = 0;

for( $i=0; $i<5; $i++ )
{
    $a += 10;
    $b += 5;
}
echo ("At the end of the loop a=$a and b=$b" );
?>
</body>
</html>
```

This will produce the following result:

```
At the end of the loop a=50 and b=25
```

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true. If the test expression is true, then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

Syntax

```
while (condition)
{
    code to be executed;
}
```

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation becomes false and the loop ends.

```
<html>
<body>
<?php
$i = 0;
$num = 50;

while( $i < 10)
{
   $num--;
   $i++;
}
echo ("Loop stopped at i = $i and num = $num" );
```

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

This will produce the following result:

```
Loop stopped at i = 10 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it will then repeat the loop as long as a condition is true.

Syntax

```
do
{
   code to be executed;
}while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable **i** as long as it has a value of less than 10:

```
<html>
<body>
<?php
$i = 0;
$num = 0;
do
{
  $i++;
}while( $i < 10 );
echo ("Loop stopped at i = $i" );
?>
</body>
</html>
```

This will produce the following result:

```
Loop stopped at i = 10
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```
foreach (array as value)
{
    code to be executed;


}
```

Example

Try out the following example to list out the values of an array.

```
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

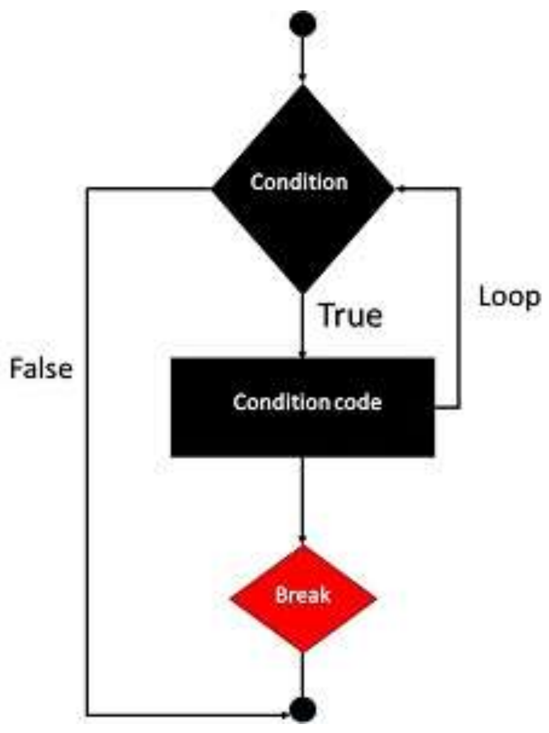
This will produce the following result:

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

The **break** statement is situated inside the statement block. If gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.



Example

In the following example, the condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
<body>

<?php
$i = 0;

while( $i < 10)
{
   $i++;
   if( $i == 3 )break;
}
echo ("Loop stopped at i = $i" );
?>
</body>
</html>
```

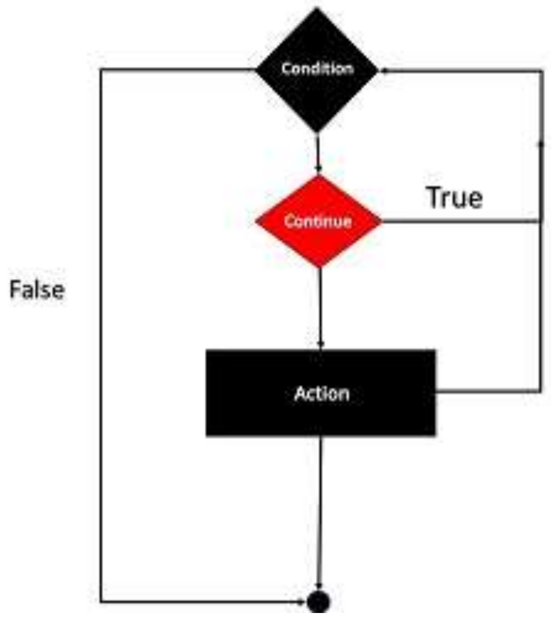
This will produce the following result:

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.



Example

In the following example, the loop prints the value of array, but when the condition becomes true, it just skips the code and next value is printed.

```
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
  if( $value == 3 )continue;
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce the following result:

```
Value is 1
Value is 2
Value is 4
Value is 5
```

9. PHP – Arrays

An array is a data structure that stores one or more similar type of values in a single value. For example, if you want to store 100 numbers, then instead of defining 100 variables, it is easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array** - An array with a numeric index. Values are stored and accessed in linear fashion
- **Associative array** - An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array** - An array containing one or more arrays and values are accessed using multiple indices

NOTE: Built-in array functions is given in function reference PHP Array Functions

Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default, the array index starts from zero.

Example

The following example demonstrates how to create and access numeric arrays.

Here we have used **array()** function to create array. This function is explained in function reference.

```
<html>
<body>
<?php
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
foreach( $numbers as $value )
{
  echo "Value is $value <br />";
}
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";
$numbers[2] = "three";
```

```
$numbers[3] = "four";
$numbers[4] = "five";


foreach( $numbers as $value )
{
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce the following result:

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five
```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE: Don't keep associative array inside double quote while printing, otherwise it would not return any value.

Example

```
<html>
<body>
<?php
/* First method to associate create array. */
```

```
$salaries = array(
               "mohammad" => 2000,
               "qadir" => 1000,
               "zara" => 500
              );

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
echo "Salary of zara is ".  $salaries['zara']. "<br />";

/* Second method to create array. */
$salaries['mohammad'] = "high";
$salaries['qadir'] = "medium";
$salaries['zara'] = "low";

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
echo "Salary of zara is ".  $salaries['zara']. "<br />";
?>
</body>
</html>
```

This will produce the following result:

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low
```

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example

In this example, we create a two dimensional array to store marks of three students in three subjects:

This example is an associative array, you can create numeric array in the same fashion.

```
<html>
<body>
<?php
   $marks = array(
            "mohammad" => array
            (
            "physics" => 35,
            "maths" => 30,
            "chemistry" => 39
            ),
            "qadir" => array
                (
                "physics" => 30,
                "maths" => 32,
                "chemistry" => 29
                ),
                "zara" => array
                (
                "physics" => 31,
                "maths" => 22,
                "chemistry" => 39
                )
          );
   /* Accessing multi-dimensional array values */
   echo "Marks for mohammad in physics : " ;
   echo $marks['mohammad']['physics'] . "<br />";
   echo "Marks for qadir in maths : ";
   echo $marks['qadir']['maths'] . "<br />";
   echo "Marks for zara in chemistry : " ;
   echo $marks['zara']['chemistry'] . "<br />";
?>
</body>
</html>
```

This will produce the following result:

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39
```

10. PHP – Strings

They are sequences of characters, like "PHP supports string operations".

NOTE: Built-in string functions is given in function reference PHP String Functions
Some valid examples of strings are as follows:

```
$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 (\)

String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator:

```
<?php
$string1="Hello World";
$string2="1234";
echo $string1 . " " . $string2;
?>
```

This will produce the following result:

```
Hello World 1234
```

If you look at the code above, you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Using the strlen() function

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!":

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

This will produce the following result:

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string).

Using the strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.
Let's see if we can find the string "world" in our string:

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

This will produce the following result:

```
 6
```

As you can see, the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

11. PHP – Web Concepts

This session demonstrates how PHP can provide dynamic content according to browser type, randomly generated numbers or User Input. It also demonstrated how the client browser can be redirected.

Identifying Browser & Platform

PHP creates some useful **environment variables** that can be seen in the phpinfo.php page that was used to setup the PHP environment.

One of the environment variables set by PHP is **HTTP_USER_AGENT** which identifies the user's browser and operating system.

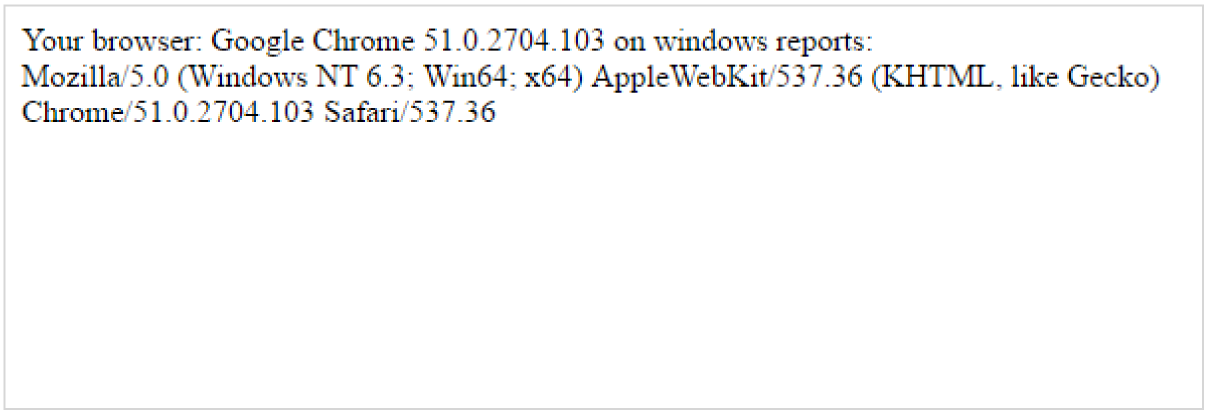
PHP provides a function getenv() to access the value of all the environment variables. The information contained in the HTTP_USER_AGENT environment variable can be used to create dynamic content appropriate to the browser.

The following example demonstrates how you can identify a client browser and operating system.

NOTE: The function preg_match()is discussed in PHP Regular expression session.

```
<html>
<body>
<?php
   $viewer = getenv( "HTTP_USER_AGENT" );
   $browser = "An unidentified browser";
   if( preg_match( "/MSIE/i", "$viewer" ) )
   {
      $browser = "Internet Explorer";
   }
   else if(  preg_match( "/Netscape/i", "$viewer" ) )
   {
      $browser = "Netscape";
   }
   else if(  preg_match( "/Mozilla/i", "$viewer" ) )
   {
      $browser = "Mozilla";
   }
   $platform = "An unidentified OS!";
   if( preg_match( "/Windows/i", "$viewer" ) )
   {
      $platform = "Windows!";
   }
   else if ( preg_match( "/Linux/i", "$viewer" ) )
   {
      $platform = "Linux!";
   }
   echo("You are using $browser on $platform");
?>
</body>
</html>
```

This is producing the following result on my machine. This result may be different for your computer, depending on what browser you are using.



```
Your browser: Google Chrome 51.0.2704.103 on windows reports:
Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/51.0.2704.103 Safari/537.36
```

Display Images Randomly

The PHP **rand()** function is used to generate a random number. This function can generate numbers with-in a given range. The random number generator should be seeded to prevent a regular pattern of numbers being generated. This is achieved using the **srand()**function that specifies the seed number as its argument.

The following example demonstrates how you can display different image each time out of four images:

```
<html>
<body>
<?php
  srand( microtime() * 1000000 );
  $num = rand( 1, 4 );

  switch( $num )
  {
  case 1: $image_file = "/home/images/alfa.jpg";
          break;
  case 2: $image_file = "/home/images/ferrari.jpg";
          break;
  case 3: $image_file = "/home/images/jaguar.jpg";
          break;
  case 4: $image_file = "/home/images/porsche.jpg";
          break;
  }
  echo "Random Image : <img src=$image_file />";
?>
</body>
</html>
```

It will produce the following result:

Using HTML Forms

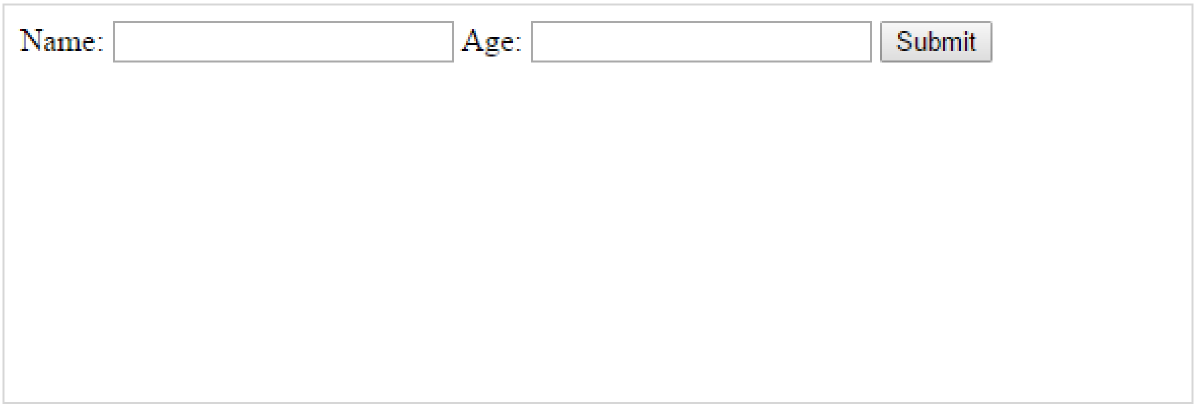
The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will automatically be available to your PHP scripts.

Try out the following example by putting the source code in test.php script.

```
<?php
  if( $_POST["name"] || $_POST["age"] )
  {
     echo "Welcome ". $_POST['name']. "<br />";
     echo "You are ". $_POST['age']. " years old.";
     exit();
  }
```

```
?>
<html>
<body>
  <form action="<?php $_PHP_SELF ?>" method="POST">
  Name: <input type="text" name="name" />
  Age: <input type="text" name="age" />
  <input type="submit" />
  </form>
</body>
</html>
```

It will produce the following result:



- The PHP default variable **$_PHP_SELF** is used for the PHP script name and when you click "submit" button, the same PHP script will be called and will produce following result:
- The method = "POST" is used to post user data to the server script. There are two methods of posting data to the server script which are discussed in PHP GET & POST chapter.

Browser Redirection

The PHP **header()** function supplies raw HTTP headers to the browser and can be used to redirect it to another location. The redirection script should be at the very top of the page to prevent any other part of the page from loading.

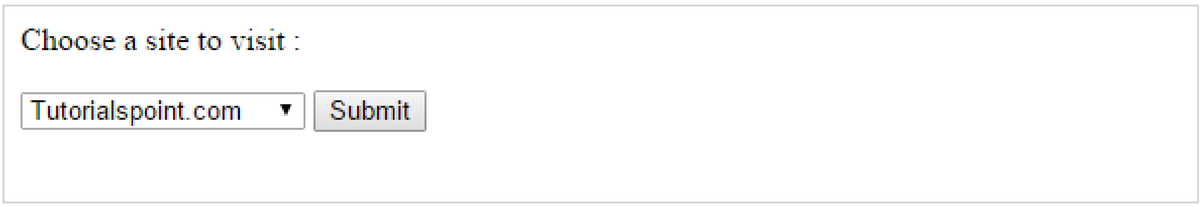
The target is specified by the **Location:** header as the argument to the **header()** function. After calling this function the **exit()** function can be used to halt parsing of rest of the code.

The following example demonstrates how you can redirect a browser request to another web page. Try out this example by putting the source code in test.php script.

```
<?php
```

```
  if( $_POST["location"] )
  {
     $location = $_POST["location"];
     header( "Location:$location" );
     exit();
  }
?>
<html>
<body>
   <p>Choose a site to visit :</p>
   <form action="<?php $_PHP_SELF ?>" method="POST">
   <select name="location">
      <option value="http://w3c.org">
            World Wise Web Consortium
      </option>
      <option value="http://www.google.com">
            Google Search Page
      </option>
   </select>
   <input type="submit" />
   </form>
</body>
</html>
```

It will produce the following output:



Displaying "File Download" Dialog Box

Sometime it is desired that you want to give option where a use will click a link and it will pop up a "File Download" box to the user instead of displaying actual content. This is very easy and will be achieved through HTTP header.
The HTTP header will be different from the actual header where we send **Content-Type** as **text/html\n\n**. In this case content type will be **application/octet-stream** and actual file name will be concatenated along with it.

For example, if you want make a **FileName** file downloadable from a given link, then its syntax will be as follows.

```
#!/usr/bin/perl

# HTTP Header
print "Content-Type:application/octet-stream; name=\"FileName\"\r\n";
print "Content-Disposition: attachment; filename=\"FileName\"\r\n\n";

# Actual File Content
open( FILE, "<FileName" );
while(read(FILE, $buffer, 100) )
{
   print("$buffer");
}
```

12. PHP – GET and POST Methods

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

Spaces are removed and replaced with the + character and any other non-alphanumeric characters are replaced with a hexadecimal values. After the information is encoded, it is sent to the server.

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** character.

```
http://www.test.com/index.htm?name1=value1&name2=value2
```

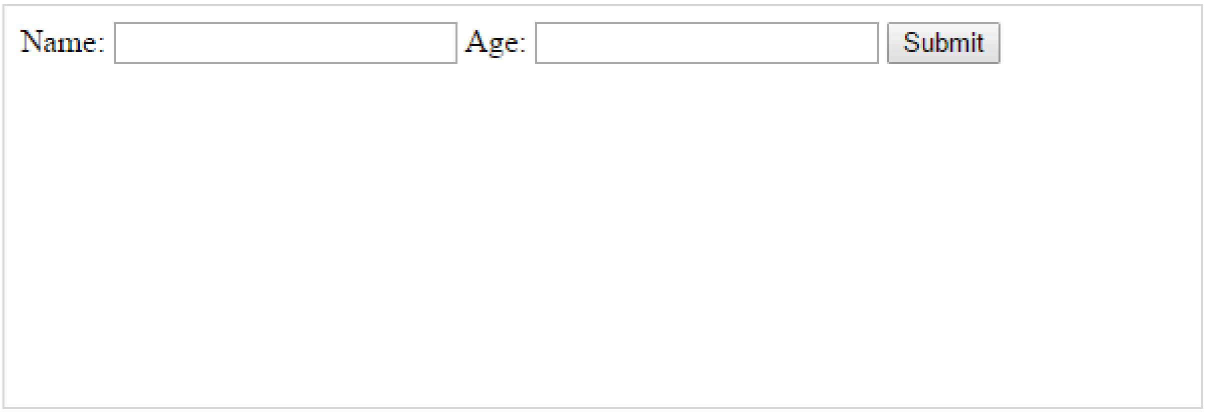
- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send up to 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **$_GET** associative array to access all the sent information using GET method.

Try out the following example by putting the source code in test.php script.

```
<?php
  if( $_GET["name"] || $_GET["age"] )
  {
     echo "Welcome ". $_GET['name']. "<br />";
     echo "You are ". $_GET['age']. " years old.";
```

```
      exit();
   }
?>
<html>
<body>
  <form action="<?php $_PHP_SELF ?>" method="GET">
  Name: <input type="text" name="name" />
  Age: <input type="text" name="age" />
  <input type="submit" />
  </form>
</body>
</html>
```

It will produce the following result:



The POST Method

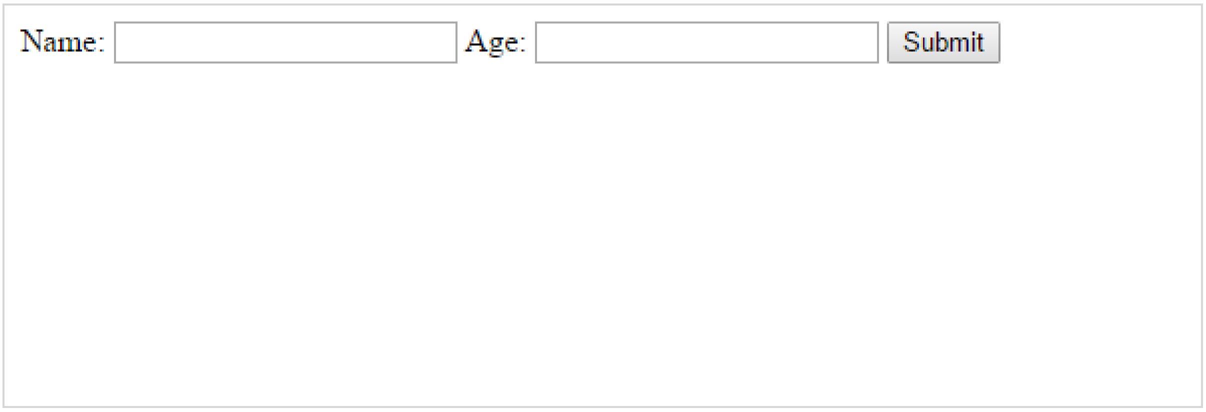
The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **$_POST** associative array to access all the sent information using POST method.

Try out the following example by putting the source code in test.php script.

```
<?php
  if( $_POST["name"] || $_POST["age"] )
  {
     echo "Welcome ". $_POST['name']. "<br />";
     echo "You are ". $_POST['age']. " years old.";
     exit();
  }
?>
<html>
<body>
  <form action="<?php $_PHP_SELF ?>" method="POST">

  Name: <input type="text" name="name" />
  Age: <input type="text" name="age" />

  <input type="submit" />
  </form>
</body>
</html>
```

It will produce the following result:



The $_REQUEST variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. We will discuss $_COOKIE variable when we will explain about cookies.

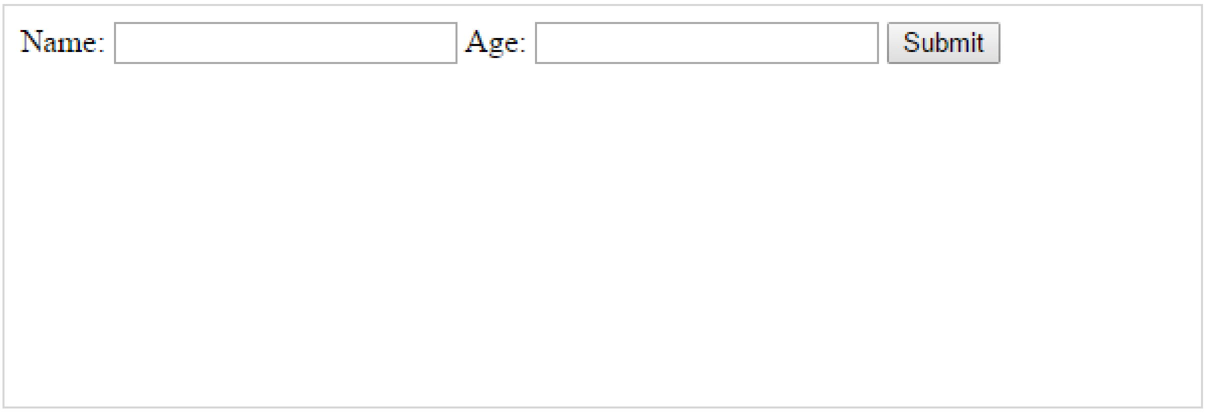
The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out the following example by putting the source code in test.php script.

```
<?php
  if( $_REQUEST["name"] || $_REQUEST["age"] )
  {
     echo "Welcome ". $_REQUEST['name']. "<br />";
     echo "You are ". $_REQUEST['age']. " years old.";
     exit();
  }
?>
<html>
<body>
  <form action="<?php $_PHP_SELF ?>" method="POST">

  Name: <input type="text" name="name" />
  Age: <input type="text" name="age" />

  <input type="submit" />
  </form>
</body>
</html>
```

Here $_PHP_SELF variable contains the name of self script in which it is being called.

It will produce the following result:



13. PHp – File Inclusion

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

- The include() Function
- The require() Function

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required, then instead of changing thousands of files just change included file.

The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file, then the **include()** function generates a warning but the script will continue execution.

Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

```
<a href="http://www.tutorialspoint.com/index.htm">Home</a> -
<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -
<a href="http://www.tutorialspoint.com/perl">PERL</a> <br />
```

Now create as many pages as you like and include this file to create header. For example now your test.php file can have the following content.

```
<html>
<body>

<?php include("menu.php"); ?>

<p>This is an example to show how to include PHP file!</p>

</body>

</html>
```

This will produce the following result:

```
Home -
ebXML -
AJAX -
PERL

This is an example to show how to include PHP file.
```

The require() Function

The require() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file, then the **require()** function generates a fatal error and halt the execution of the script.

So there is no difference in require() and include() except they handle error conditions. It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

You can try using above example with require() function and it will generate same result. But if you will try the following two examples where file does not exist, then you will get different results.

```
<html>
<body>
<?php include("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This will produce the following result:

```
This is an example to show how to include wrong PHP file!
```

Now let us try same example with require() function.

```
<html>

<body>

<?php require("xxmenu.php"); ?>

<p>This is an example to show how to include wrong PHP file!</p>

</body>

</html>
```

This time file execution halts and nothing is displayed.

NOTE: You may get plain warning messages or fatal error messages or nothing at all. This depends on your PHP Server configuration.

14. PHP – Files & I/O

This chapter will explain the following functions related to files:

- Opening a file
- Reading a file
- Writing a file
- Closing a file

Opening and Closing Files

The PHP **fopen()** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Mode	Purpose
r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If the file does not exist, then it attempts to create a file.
w+	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If the file does not exist, then it attempts to create a file.
a	Opens the file for writing only. Places the file pointer at the end of the file. If the file does not exist, then it attempts to create a file.
a+	Opens the file for reading and writing only. Places the file pointer at the end of the file. If the file does not exist, then it attempts to create a file.

If an attempt to open a file fails, then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.

,

After making a changes to the opened file it is important to close it with the **fclose()**function. The **fclose()** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The file's length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.
- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.

The following example assigns the content of a text file to a variable and then displays those contents on the web page.

```
<html>
<head>
<title>Reading a file using PHP</title>
</head>
<body>

<?php
$filename = "/home/user/guest/tmp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
   echo ( "Error in opening file" );
   exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );

fclose( $file );

echo ( "File size : $filesize bytes" );
echo ( "<pre>$filetext</pre>" );
?>


</body>
</html>
```

It will produce the following result:

```
File size : 278 bytes

The PHP Hypertext Preprocessor (PHP) is a programming
language that allows web developers to create dynamic
content that interacts with databases.
PHP is basically used for developing web based software
applications. This tutorial helps you to build your base
 with PHP.
```

Writing a File

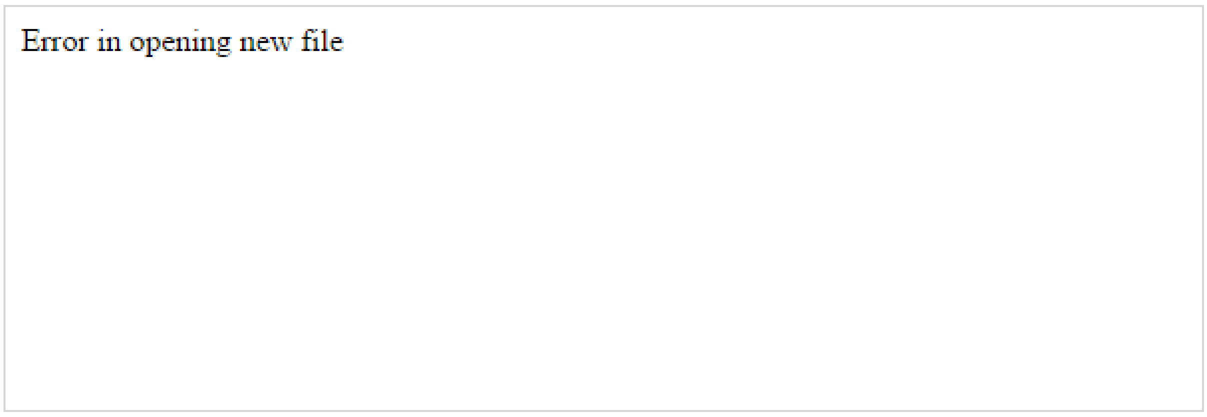
A new file can be written or text can be appended to an existing file using the PHP **fwrite()**function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would will stop after the specified length has been reached.

The following example creates a new text file and then writes a short text heading inside it. After closing this file its existence is confirmed using **file_exist()** function which takes file name as an argument

```
<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
   echo ( "Error in opening new file" );
   exit();
}
fwrite( $file, "This is  a simple test\n" );
fclose( $file );
?>
```

```
<html>
<head>
<title>Writing a file using PHP</title>
</head>
<body>

<?php
if( file_exist( $filename ) )
{
   $filesize = filesize( $filename );
   $msg = "File  created with name $filename ";
   $msg .= "containing $filesize bytes";
   echo ($msg );
}
else
{
   echo ("File $filename does not exit" );
}
?>
</body>
</html>
```

It will produce the following result:



```
Error in opening new file
```

We have covered all the function related to file input and out in the PHP File System Function chapter.

15. PHP – Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you:

- Creating a PHP Function
- Calling a PHP Function

In fact, you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Please refer to PHP Function Reference for a complete set of useful functions.

Creating PHP Function

It is very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it.

The following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below:

```
<html>
<head>
<title>Writing PHP Function</title>
</head>
<body>

<?php
/* Defining a PHP Function */
function writeMessage()
{
  echo "You are really a nice person, Have a nice time!";
}
/* Calling a PHP Function */
```

```
writeMessage();
?>
</body>
</html>
```

This will display the following result:

```
You are really a nice person, Have a nice time!
```

PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. The following example takes two integer parameters and adds them together and then prints them.

```
<html>
<head>
<title>Writing PHP Function with Parameters</title>
</head>
<body>

<?php
function addFunction($num1, $num2)
{
  $sum = $num1 + $num2;
  echo "Sum of the two numbers is : $sum";
}
addFunction(10, 20);
?>
</body>
</html>
```

This will display the following result:

```
Sum of the two numbers is : 30
```

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

The following example depicts both the cases.

```
<html>
<head>
<title>Passing Argument by Reference</title>
</head>
<body>
<?php
function addFive($num)
{
   $num += 5;
}

function addSix(&$num)
{
   $num += 6;
}
$orignum = 10;
addFive( &$orignum );
echo "Original Value is $orignum<br />";
addSix( $orignum );
echo "Original Value is $orignum<br />";
?>
</body>
</html>
```

This will display the following result:

```
Original Value is 10
Original Value is 16
```

PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using **return array(1,2,3,4)**.

The following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html>

<head>

<title>Writing PHP Function which returns value</title>

</head>

<body>
```

```
<?php
function addFunction($num1, $num2)
{
  $sum = $num1 + $num2;
  return $sum;
}
$return_value = addFunction(10, 20);
echo "Returned value from the function : $return_value";
?>
</body>
</html>
```

This will display the following result:

```
Returned value from the function : 30
```

Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it.

The following function prints NULL in case use does not pass any value to this function.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>

<?php
function printMe($param = NULL)
{
   print $param;
}
printMe("This is test");
printMe();
?>

</body>
</html>
```

This will produce the following result:

```
This is test
```

Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. The following example depicts this behavior.

```
<html>
<head>
<title>Dynamic Function Calls</title>
</head>
<body>
<?php
function sayHello()
{
   echo "Hello<br />";
}
$function_holder = "sayHello";
$function_holder();
?>
</body>
</html>
```

This will display the following result:

```
Hello
```

16. PHP – Cookies

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users:

- Server script sends a set of cookies to the browser. For example, name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- Next time, when the browser sends any request to the web server, it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set cookies, how to access them and how to delete them.

The Anatomy of a Cookie

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A PHP script that sets a cookie might send headers that look something like this:

```
HTTP/1.1 200 OK

Date: Fri, 04 Feb 2000 21:03:38 GMT

Server: Apache/1.3.9 (UNIX) PHP/4.0b3

Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;

                 path=/; domain=tutorialspoint.com

Connection: close

Content-Type: text/html
```

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this:

```
GET / HTTP/1.0

Connection: Keep-Alive

User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)

Host: zink.demon.co.uk:1126
```

```
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

A PHP script will then have access to the cookie in the environmental variables $_COOKIE or $HTTP_COOKIE_VARS[] which holds all cookie names and values. Above cookie can be accessed using $HTTP_COOKIE_VARS["name"].

Setting Cookies with PHP

PHP provided **setcookie()** function to set a cookie. This function requires up to six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

```
setcookie(name, value, expire, path, domain, security);
```

Here is the detail of all the arguments:

- **Name -** This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value -**This sets the value of the named variable and is the content that you actually want to store.
- **Expiry -** This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set, then cookie will automatically expire when the Web Browser is closed.
- **Path -**This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain -** This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security -** This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

The following example will create two cookies **name** and **age**. These cookies will expire after an hour.

```
<?php
   setcookie("name", "John Watkin", time()+3600, "/","", 0);
   setcookie("age", "36", time()+3600, "/", "",  0);
?>
<html>
<head>
```

```
<title>Setting Cookies with PHP</title>
</head>
<body>
<?php echo "Set Cookies"?>
</body>
</html>
```

Accessing Cookies with PHP

PHP provides many ways to access cookies. The simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. The following example will access all the cookies set in above example.

```
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
echo $_COOKIE["name"]. "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"]. "<br />";

echo $_COOKIE["age"] . "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"] . "<br />";
?>
</body>
</html>
```

You can use **isset()** function to check if a cookie is set or not.

```
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
  if( isset($_COOKIE["name"]))
    echo "Welcome " . $_COOKIE["name"] . "<br />";
  else
```

```
    echo "Sorry... Not recognized" . "<br />";
?>
</body>
</html>
```

Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired:

```
<?php
  setcookie( "name", "", time()- 60, "/","", 0);
  setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>
<head>
<title>Deleting Cookies with PHP</title>
</head>
<body>
<?php echo "Deleted Cookies" ?>
</body>
</html>
```

17. PHP – Sessions

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the **php.ini** file called **session.save_path**. Bore using any session variable make sure you have setup this path. When a session is started, the following actions take place:

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.

- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.

- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the **session_start()** function. This function first checks if a session is already started and if none is started, then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page.

Session variables are stored in associative array called **$_SESSION[]**. These variables can be accessed during lifetime of a session.

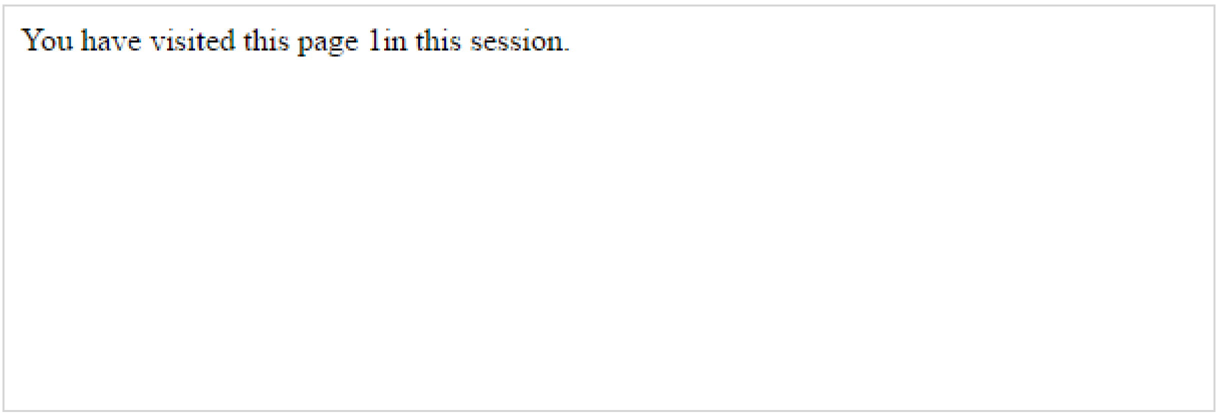
The following example starts a session and then registers a variable called **counter** that is incremented each time the page is visited during the session.

Make use of **isset()** function to check if session variable is already set or not.

Put this code in a test.php file and load this file many times to see the result:

```
<?php
   session_start();
   if( isset( $_SESSION['counter'] ) )
   {
      $_SESSION['counter'] += 1;
   }
   else
   {
      $_SESSION['counter'] = 1;
   }
   $msg = "You have visited this page ".  $_SESSION['counter'];
   $msg .= "in this session.";
?>
<html>
<head>
<title>Setting up a PHP session</title>
</head>
<body>
<?php  echo ( $msg ); ?>
</body>
</html>
```

It will produce the following result:



```
You have visited this page 1in this session.
```

Destroying a PHP Session

A PHP session can be destroyed by **session_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable, then you can use **unset()** function to unset a session variable.

Here is the example to unset a single variable:

```
<?php
   unset($_SESSION['counter']);
?>
```

Here is the call which will destroy all the session variables:

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

Turning on Auto Session

You don't need to call start_session() function to start a session when a user visits your site if you can set **session.auto_start** variable to 1 in **php.ini** file.

Sessions without cookies

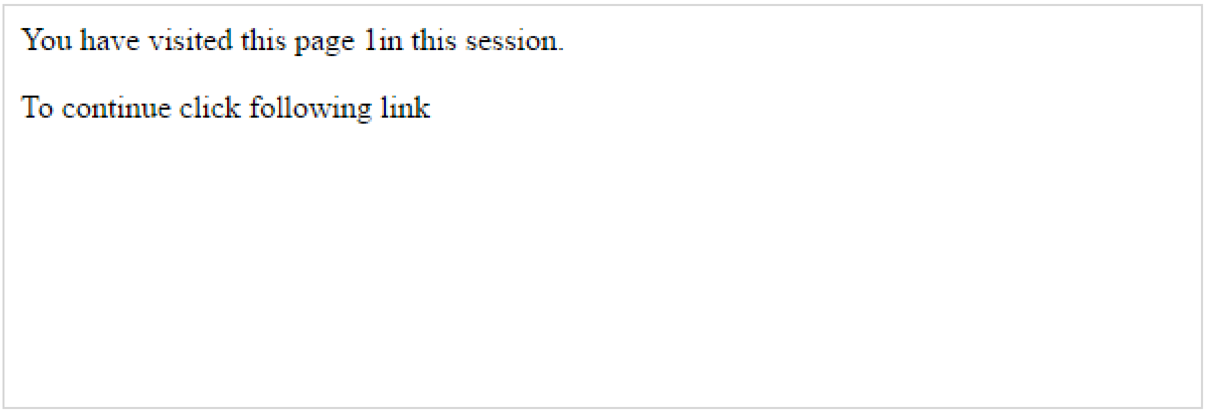
There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

```
<?php
   session_start();

   if (isset($_SESSION['counter'])) {
      $_SESSION['counter'] = 1;
   } else {
      $_SESSION['counter']++;
   }
?>
   $msg = "You have visited this page ".  $_SESSION['counter'];
   $msg .= "in this session.";
   echo ( $msg );
<p>
To continue  click following link <br />
<a  href="nextpage.php?<?php echo htmlspecialchars(SID); >">
</p>
```

It will produce the following result:



```
You have visited this page 1in this session.

To continue click following link
```

The **htmlspecialchars()** may be used when printing the SID in order to prevent XSS related attacks.

18. PHP – Sending Emails

PHP must be configured correctly in the **php.ini** file with the details of how your system sends email. Open php.ini file available in **/etc/** directory and find the section headed **[mail function]**.

Windows users should ensure that two directives are supplied. The first is called SMTP that defines your email server address. The second is called sendmail_from which defines your own email address.

The configuration for Windows should look something like this:

```
[mail function]
; For Win32 only.
SMTP = smtp.secureserver.net

; For win32 only
sendmail_from = webmaster@tutorialspoint.com
```

Linux users simply need to let PHP know the location of their sendmail application. The path and any desired switches should be specified to the sendmail_path directive.

The configuration for Linux should look something like this:

```
[mail function]
; For Win32 only.
SMTP =


; For win32 only
sendmail_from =


; For Unix only
sendmail_path = /usr/sbin/sendmail -t -i
```

Now you are ready to go.

Sending plain text email

PHP makes use of **mail()** function to send an email. This function requires three mandatory arguments that specify the recipient's email address, the subject of the message and the actual message additionally there are other two optional parameters.

```
mail( to, subject, message, headers, parameters );
```

Here is the description for each parameters.

Parameter	Description
to	Required. Specifies the receiver / receivers of the email
subject	Required. Specifies the subject of the email. This parameter cannot contain any newline characters
message	Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters
headers	Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n)
parameters	Optional. Specifies an additional parameter to the sendmail program

As soon as the mail function is called, PHP will attempt to send the email, then it will return true if successful or false if it is failed.

Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.

Example

The following example will send an HTML email message to xyz@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

```
<html>
<head>
<title>Sending email using PHP</title>
</head>
<body>
<?php
   $to = "xyz@somedomain.com";
   $subject = "This is subject";
   $message = "This is simple text message.";
   $header = "From:abc@somedomain.com \r\n";
   $retval = mail ($to,$subject,$message,$header);
   if( $retval == true )
   {
      echo "Message sent successfully...";
   }
   else
   {
      echo "Message could not be sent...";
   }
?>
</body>
</html>
```

Sending HTML email

When you send a text message using PHP, then all the content will be treated as simple text. Even if you will include HTML tags in a text message, it will be displayed as simple text and HTML tags will not be formatted according to HTML syntax. But PHP provides option to send an HTML message as actual HTML message.

While sending an email message, you can specify a Mime version, content type and character set to send an HTML email.

Example

The following example will send an HTML email message to xyz@somedomain.com copying it to afgh@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

```
<html>
<head>
<title>Sending HTML email using PHP</title>
</head>
<body>
<?php
   $to = "xyz@somedomain.com";
   $subject = "This is subject";
   $message = "<b>This is HTML message.</b>";
   $message .= "<h1>This is headline.</h1>";
   $header = "From:abc@somedomain.com \r\n";
   $header = "Cc:afgh@somedomain.com \r\n";
   $header .= "MIME-Version: 1.0\r\n";
   $header .= "Content-type: text/html\r\n";
   $retval = mail ($to,$subject,$message,$header);
   if( $retval == true )
   {
      echo "Message sent successfully...";
   }
   else
   {
      echo "Message could not be sent...";
   }
?>
</body>
</html>
```

Sending attachments with email

To send an email with mixed content requires to set **Content-type** header to **multipart/mixed**. Then text and attachment sections can be specified within **boundaries**.

A boundary is started with two hyphens followed by a unique number which can not appear in the message part of the email. A PHP function **md5()** is used to create a 32 digit hexadecimal number to create unique number. A final boundary denoting the email's final section must also end with two hyphens.

Attached files should be encoded with the **base64_encode()** function for safer transmission and are best split into chunks with the **chunk_split()** function. This adds **\r\n** inside the file at regular intervals, normally every 76 characters.

Following is the example which will send a file **/tmp/test.txt** as an attachment. you can code your program to receive an uploaded file and send it.

```
<html>
<head>
<title>Sending attachment using PHP</title>
</head>
<body>
<?php
  $to = "xyz@somedomain.com";
  $subject = "This is subject";
  $message = "This is test message.";
  # Open a file
  $file = fopen( "/tmp/test.txt", "r" );
  if( $file == false )
  {
     echo "Error in opening file";
     exit();
  }
  # Read the file into a variable
  $size = filesize("/tmp/test.txt");
  $content = fread( $file, $size);
```

```
# encode the data for safe transit
# and insert \r\n after every 76 chars.
$encoded_content = chunk_split( base64_encode($content));

# Get a random 32 bit number using time() as seed.
$num = md5( time() );

# Define the main headers.
$header = "From:xyz@somedomain.com\r\n";
$header .= "MIME-Version: 1.0\r\n";
$header .= "Content-Type: multipart/mixed; ";
$header .= "boundary=$num\r\n";
$header .= "--$num\r\n";

# Define the message section
$header .= "Content-Type: text/plain\r\n";
$header .= "Content-Transfer-Encoding:8bit\r\n\n";
$header .= "$message\r\n";
$header .= "--$num\r\n";

# Define the attachment section
$header .= "Content-Type:  multipart/mixed; ";
$header .= "name=\"test.txt\"\r\n";
$header .= "Content-Transfer-Encoding:base64\r\n";
$header .= "Content-Disposition:attachment; ";
$header .= "filename=\"test.txt\"\r\n\n";
$header .= "$encoded_content\r\n";
$header .= "--$num--";

# Send email now
$retval = mail ( $to, $subject, "", $header );
if( $retval == true )
 {
    echo "Message sent successfully...";
 }
 else
 {
      echo "Message could not be sent...";
   }
?>
</body>
</html>
```

You try all the above examples. If you face any problems, then you can post it in our discussion forum.

19. PHP – File Uploading

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload_tmp_dir** and the maximum permitted size of files that can be uploaded is stated as **upload_max_filesize**. These parameters are set into PHP configuration file**php.ini**

The process of uploading a file follows these steps:

- The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text field, then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual, while writing files, it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only, then process will fail.

An uploaded file could be a text file or image file or any document.

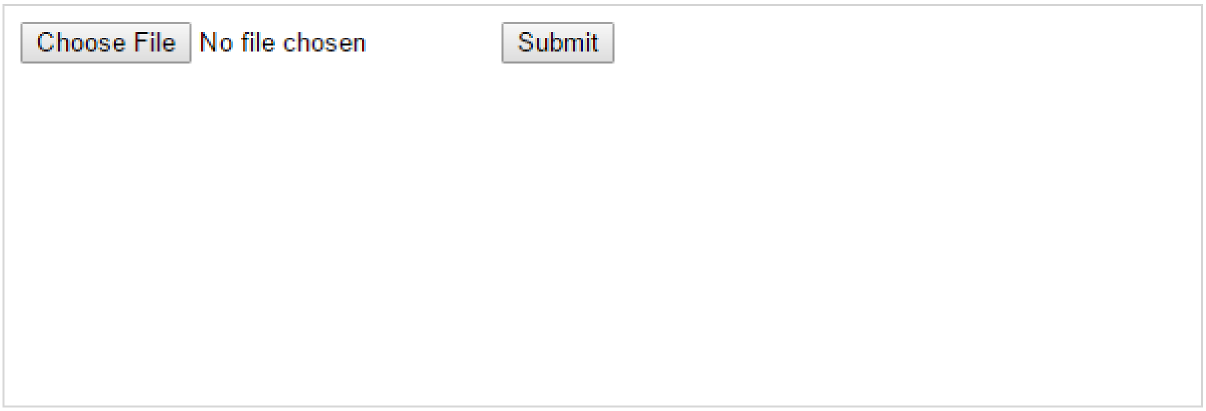
Creating an Upload Form

The following HTML code creates an uploader form. This form is having method attribute set to **post** and enctype attribute is set to **multipart/form-data**

```
<html>


<head>
<title>File Uploading Form</title>
</head>
<body>
<h3>File Upload:</h3>
Select a file to upload: <br />
```

```
<form action="/php/file_uploader.php" method="post"
                        enctype="multipart/form-data">
<input type="file" name="file" size="50" />
<br />
<input type="submit" value="Upload File" />
</form>
</body>
</html>
```

This will display the following result:



Creating an upload script

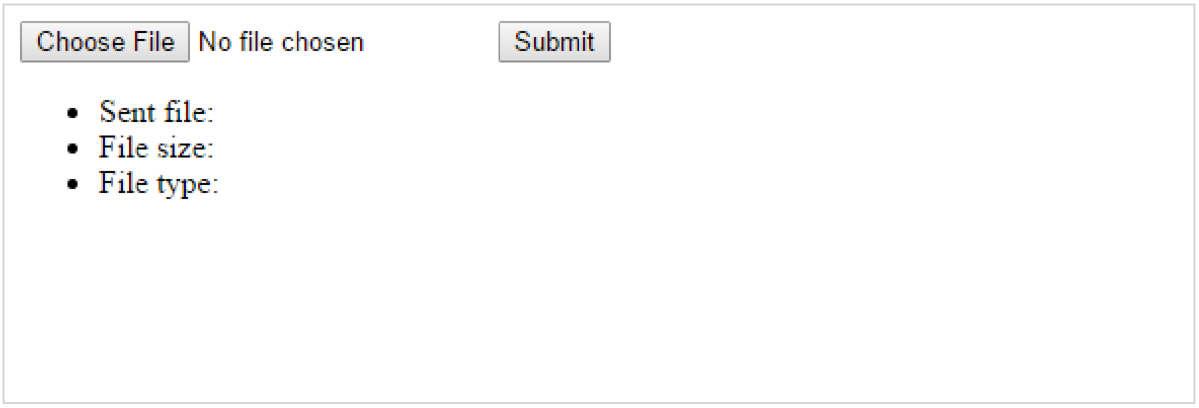
There is one global PHP variable called **$_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create the following five variables:

- **$_FILES['file']['tmp_name']-** the uploaded file in the temporary directory on the web server.
- **$_FILES['file']['name'] -** the actual name of the uploaded file.
- **$_FILES['file']['size'] -** the size in bytes of the uploaded file.
- **$_FILES['file']['type'] -** the MIME type of the uploaded file.
- **$_FILES['file']['error'] -** the error code associated with this file upload.

The following example should allow upload images and return the uploaded file information.

```
<?php
if( $_FILES['file']['name'] != "" )
{
   copy( $_FILES['file']['name'], "/var/www/html" ) or
           die( "Could not copy file!");
}
else
{
    die("No file specified!");
}
?>
<html>
<head>
<title>Uploading Complete</title>
</head>
<body>
<h2>Uploaded File Info:</h2>
<ul>
<li>Sent file: <?php echo $_FILES['file']['name'];  ?>
<li>File size: <?php echo $_FILES['file']['size'];  ?> bytes
<li>File type: <?php echo $_FILES['file']['type'];  ?>
</ul>
</body>
</html>
```

It will produce the following result:



- Sent file:
- File size:
- File type:

20. PHP – Coding Standard

Every company follows a different coding standard based on their best practices. Coding standard is required because there may be many developers working on different modules. If they start inventing their own standards, then the source will become very un-manageable and it will become difficult to maintain that source code in future.

Here are several reasons why to use coding specifications:

- Your peer programmers have to understand the code you produce. A coding standard acts as the blueprint for all the team to decipher the code.

- Simplicity and clarity achieved by consistent coding saves you from common mistakes.

- If you revise your code after some time, then it becomes easy to understand that code.

There are a few guidelines which can be followed while coding in PHP.

- **Indenting and Line Length** - Use an indent of 4 spaces and don't use any tab because different computers use different setting for tab. It is recommended to keep lines at approximately 75-85 characters long for better code readability.

- **Control Structures** - These include if, for, while, switch, etc. Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls. You are strongly encouraged to always use curly braces even in situations where they are technically optional.

Example

```
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    default action;
}
```

You can write switch statements as follows:

```
switch (condition) {
case 1:
    action1;
    break;
case 2:
    action2;
    break;

default:
    defaultaction;
    break;
}
```

Function Calls - Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

```
$var = foo($bar, $baz, $quux);
```

Function Definitions - Function declarations follow the "BSD/Allman style":

```
function fooFunction($arg1, $arg2 = '')
{
    if (condition) {
        statement;
    }
    return $val;
}
```

Comments - C style comments (/* */) and standard C++ comments (//) are both fine. Use of Perl/shell style comments (#) is discouraged.

PHP Code Tags - Always use <?php ?> to delimit PHP code, not the <? ?> shorthand. This is required for PHP compliance and is also the most portable way to include PHP code on differing operating systems and setups.

Variable Names -

- Use all lower case letters
- Use '_' as the word separator.
- Global variables should be prepended with a 'g'.
- Global constants should be all caps with '_' separators.
- Static variables may be prepended with 's'.

Make Functions Reentrant - Functions should not keep static variables that prevent a function from being reentrant.

Alignment of Declaration Blocks - Block of declarations should be aligned.

One Statement Per Line - There should be only one statement per line unless the statements are very closely related.

Short Methods or Functions - Methods should limit themselves to a single page of code. There could be many more points which should be considered while writing your PHP program. Over all intension should be to be consistent throughout of the code programming and it will be possible only when you will follow any coding standard. You can device your own standard if you like something different.

Part 2: Advanced PHP

21. PHP – Predefined Variables

PHP provides a large number of predefined variables to any script which it runs. PHP provides an additional set of predefined arrays containing variables from the web server the environment, and user input. These new arrays are called superglobals:

All the following variables are automatically available in every scope.

PHP Superglobals

Variable	Description
$GLOBALS	Contains a reference to every variable which is currently available within the global scope of the script. The keys of this array are the names of the global variables.
$_SERVER	This is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server. There is no guarantee that every web server will provide any of these. See next section for a complete list of all the SERVER variables.
$_GET	An associative array of variables passed to the current script via the HTTP GET method.
$_POST	An associative array of variables passed to the current script via the HTTP POST method.
$_FILES	An associative array of items uploaded to the current script via the HTTP POST method.
$_REQUEST	An associative array consisting of the contents of $_GET, $_POST, and $_COOKIE.
$_COOKIE	An associative array of variables passed to the current script via HTTP cookies.
$_SESSION	An associative array containing session variables available to the current script.
$_PHP_SELF	A string containing PHP script file name in which it is called.
$php_errormsg	$php_errormsg is a variable containing the text of the last error message generated by PHP.

Server variables: $_SERVER

$_SERVER is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server. There is no guarantee that every web server will provide any of these.

Variable	Description
$_SERVER['PHP_SELF']	The filename of the currently executing script, relative to the document root
$_SERVER['argv']	Array of arguments passed to the script. When the script is run on the command line, this gives C-style access to the command line parameters. When called via the GET method, this will contain the query string.
$_SERVER['argc']	Contains the number of command line parameters passed to the script if run on the command line.
$_SERVER['GATEWAY_INTERFACE']	What revision of the CGI specification the server is using; i.e. 'CGI/1.1'.
$_SERVER['SERVER_ADDR']	The IP address of the server under which the current script is executing.
$_SERVER['SERVER_NAME']	The name of the server host under which the current script is executing. If the script is running on a virtual host, this will be the value defined for that virtual host.
$_SERVER['SERVER_SOFTWARE']	Server identification string, given in the headers when responding to requests.
$_SERVER['SERVER_PROTOCOL']	Name and revision of the information protocol via which the page was requested; i.e. 'HTTP/1.0';
$_SERVER['REQUEST_METHOD']	Which request method was used to access the page; i.e. 'GET', 'HEAD', 'POST', 'PUT'.
$_SERVER['REQUEST_TIME']	The timestamp of the start of the request. Available since PHP 5.1.0.
$_SERVER['QUERY_STRING']	The query string, if any, via which the page was accessed.
$_SERVER['DOCUMENT_ROOT']	The document root directory under which the current script is executing, as defined in the server's configuration file.

$_SERVER['HTTP_ACCEPT']	Contents of the Accept: header from the current request, if there is one.
$_SERVER['HTTP_ACCEPT_CHARSET']	Contents of the Accept-Charset: header from the current request, if there is one. Example: 'iso-8859-1,*,utf-8'.
$_SERVER['HTTP_ACCEPT_ENCODING']	Contents of the Accept-Encoding: header from the current request, if there is one. Example: 'gzip'.
$_SERVER['HTTP_ACCEPT_LANGUAGE']	Contents of the Accept-Language: header from the current request, if there is one. Example: 'en'.
$_SERVER['HTTP_CONNECTION']	Contents of the Connection: header from the current request, if there is one. Example: 'Keep-Alive'.
$_SERVER['HTTP_HOST']	Contents of the Host: header from the current request, if there is one.
$_SERVER['HTTP_REFERER']	The address of the page (if any) which referred the user agent to the current page.
$_SERVER['HTTP_USER_AGENT']	This is a string denoting the user agent being which is accessing the page. A typical example is: Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586).
$_SERVER['HTTPS']	Set to a non-empty value if the script was queried through the HTTPS protocol.
$_SERVER['REMOTE_ADDR']	The IP address from which the user is viewing the current page.
$_SERVER['REMOTE_HOST']	The Host name from which the user is viewing the current page. The reverse dns lookup is based off the REMOTE_ADDR of the user.
$_SERVER['REMOTE_PORT']	The port being used on the user's machine to communicate with the web server.
$_SERVER['SCRIPT_FILENAME']	The absolute pathname of the currently executing script.
$_SERVER['SERVER_ADMIN']	The value given to the SERVER_ADMIN (for Apache) directive in the web server configuration file.
$_SERVER['SERVER_PORT']	The port on the server machine being used by the web server for communication. For default setups, this will be '80'.

$_SERVER['SERVER_SIGNATURE']	String containing the server version and virtual host name which are added to server-generated pages, if enabled.
$_SERVER['PATH_TRANSLATED']	Filesystem based path to the current script.
$_SERVER['SCRIPT_NAME']	Contains the current script's path. This is useful for pages which need to point to themselves.
$_SERVER['REQUEST_URI']	The URI which was given in order to access this page; for instance, '/index.html'.
$_SERVER['PHP_AUTH_DIGEST']	When running under Apache as module doing Digest HTTP authentication this variable is set to the 'Authorization' header sent by the client.
$_SERVER['PHP_AUTH_USER']	When running under Apache or IIS (ISAPI on PHP 5) as module doing HTTP authentication this variable is set to the username provided by the user.
$_SERVER['PHP_AUTH_PW']	When running under Apache or IIS (ISAPI on PHP 5) as module doing HTTP authentication this variable is set to the password provided by the user.
$_SERVER['AUTH_TYPE']	When running under Apache as module doing HTTP authenticated this variable is set to the authentication type.

22. PHP – Regular Expression

Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.

Using regular expression you can search a particular string inside another string, you can replace one string by another string and you can split a string into many chunks.

PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. You can use any of them based on your comfort.

- POSIX Regular Expressions
- PERL Style Regular Expressions

POSIX Regular Expressions

The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.

The simplest regular expression is one that matches a single character, such as g, inside strings such as g, haggle, or bag.

Let us discuss a few concepts being used in POSIX regular expression. After that, we will introduce you to regular expression related functions.

Brackets

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Expression	Description
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, {int. range}, and $ flags all follow a character sequence.

Expression	Description
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing zero or more p's. This is just an alternative way to use p*.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

Examples

Following examples will clear your concepts about matching characters.

Expression	Description
[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.
p.p	It matches any string containing p, followed by any character, in turn followed by another p.
^.{2}$	It matches any string containing exactly two characters.
<b>(.*)</b>	It matches any string enclosed within <b> and </b>.
p(hp)*	It matches any string containing a p followed by zero or more instances of the sequence hp.

Predefined Character Ranges

For your programming convenience several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set:

Expression	Description
[[:alpha:]]	It matches any string containing alphabetic characters aA through zZ.
[[:digit:]]	It matches any string containing numerical digits 0 through 9.
[[:alnum:]]	It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
[[:space:]]	It matches any string containing a space.

PHP's Regexp POSIX Functions

PHP currently offers seven functions for searching strings using POSIX-style regular expressions:

Function	Description
ereg()	The ereg() function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise.
ereg_replace()	The ereg_replace() function searches for string specified by pattern and replaces pattern with replacement if found.
eregi()	The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive.
eregi_replace()	The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.
split()	The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
spliti()	The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive.
sql_regcase()	The sql_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

PHP – Function ereg()

Syntax

```
int ereg(string pattern, string originalstring, [array regs]);
```

Definition and Usage

The ereg() function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise. The search is case sensitive in regard to alphabetical characters.

The optional input parameter regs contains an array of all matched expressions that were grouped by parentheses in the regular expression.

Return Value

- It returns true if the pattern is found, and false otherwise.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $email_id = "admin@tutorialspoint.com";
   $retval = ereg("(\.)(com$)", $email_id);

   if( $retval == true )
   {
      echo "Found a .com<br>";
   }
   else
   {
      echo "Could not found a .com<br>";
   }

   $retval = ereg(("(\.)(com$)"), $email_id, $regs);

   if( $retval == true )
   {
      echo "Found a .com and reg = ". $regs[0];
   }
   else
   {
      echo "Could not found a .com";
   }

?>
```

This will produce the following result –

```
Found a .com
Found a .com and reg = .com
```

PHP – Function ereg_replace()

Syntax

```
string ereg_replace (string pattern, string replacement, string
originalstring);
```

Definition and Usage

The ereg_replace() function searches for string specified by pattern and replaces pattern with replacement if found. The ereg_replace() function operates under the same premises as ereg(), except that the functionality is extended to finding and replacing pattern instead of simply locating it.

Like ereg(), ereg_replace() is case sensitive.

Return Value

- After the replacement has occurred, the modified string will be returned.
- If no matches are found, the string will remain unchanged.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $copy_date = "Copyright 1999";
   $copy_date = ereg_replace("([0-9]+)", "2000", $copy_date);

   print $copy_date;
?>
```

This will produce the following result –

```
Copyright 2000
```

PHP – Function eregi()

Syntax

```
int eregi(string pattern, string string, [array regs]);
```

Definition and Usage

The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive. Eregi() can be particularly useful when checking the validity of strings, such as passwords.

The optional input parameter regs contains an array of all matched expressions that were grouped by parentheses in the regular expression.

Return Value

- It returns true if the pattern is validated, and false otherwise.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $password = "abc";

   if (! eregi ("[[:alnum:]]{8,10}", $password))
   {
      print "Invalid password! Passwords must be from 8 - 10 chars";
   }
   else
   {
      print "Valid password";
   }
?>
```

This will produce the following result –

```
Invalid password! Passwords must be from 8 - 10 chars
```

PHP – Function eregi_replace()

Syntax

```
string eregi_replace (string pattern, string replacement, string
originalstring);
```

Definition and Usage

The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.

Return Value

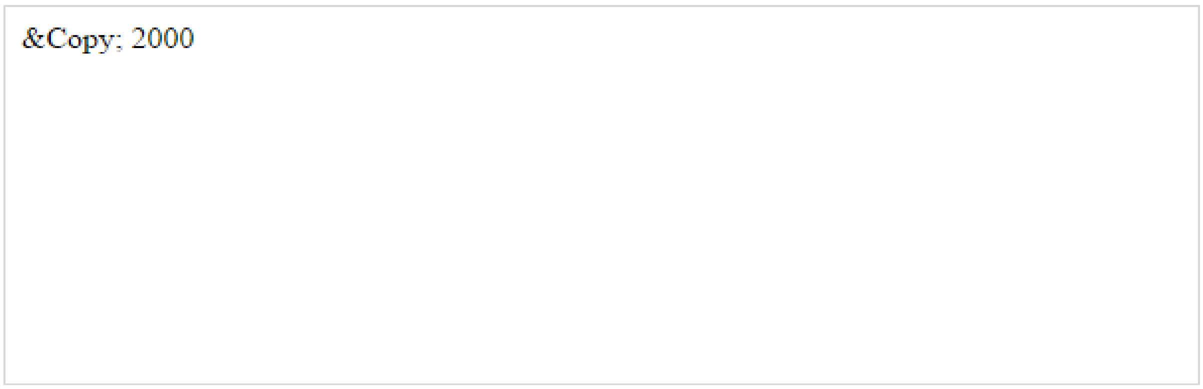
- After the replacement has occurred, the modified string will be returned.
- If no matches are found, the string will remain unchanged.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $copy_date = "Copyright 2000";
   $copy_date = eregi_replace("([a-z]+)", "&Copy;", $copy_date);

   print $copy_date;
?>
```

This will produce the following result –



```
&Copy; 2000
```

PHP – Function split()

Syntax

```
array split (string pattern, string string [, int limit])
```

Definition and Usage

The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.

The optional input parameter limit is used to signify the number of elements into which the string should be divided, starting from the left end of the string and working rightward. In cases where the pattern is an alphabetical character, split() is case sensitive.

Return Value

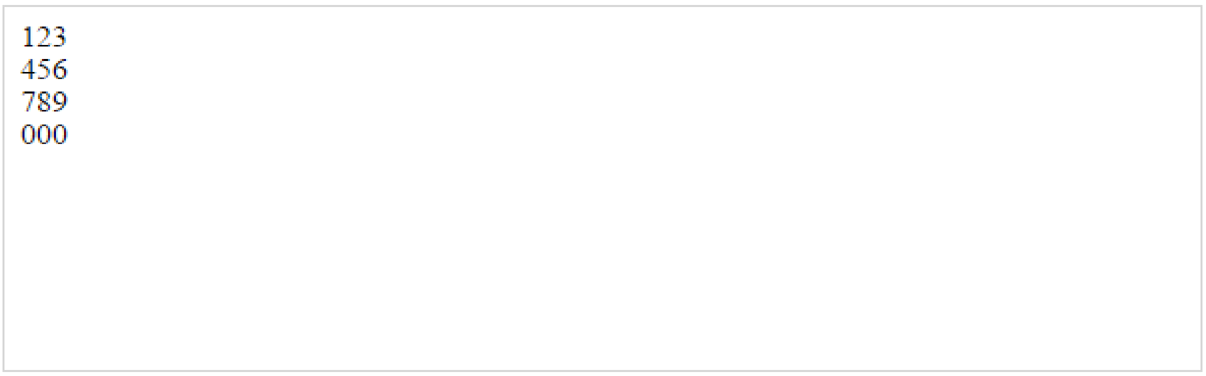
- Returns an array of strings after splitting up a string.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php


   $ip = "123.456.789.000"; // some IP address
   $iparr = split ("\.", $ip);


   print "$iparr[0] <br />";
   print "$iparr[1] <br />" ;
   print "$iparr[2] <br />"  ;
   print "$iparr[3] <br />"  ;


?>
```

This will produce the following result –



```
123
456
789
000
```

PHP – Function spliti()

Syntax

```
array spliti (string pattern, string string [, int limit])
```

Definition and Usage

The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive. Case-sensitive characters are an issue only when the pattern is alphabetical. For all other characters, spliti() operates exactly as split() does.

Return Value

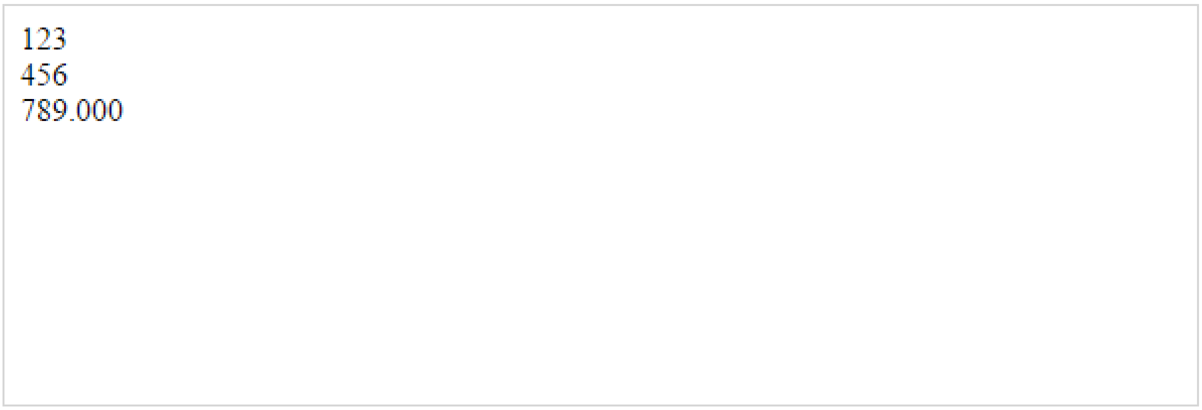
- Returns an array of strings after splitting up a string.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $ip = "123.456.789.000"; // some IP address
   $iparr = spliti ("\.", $ip, 3);

   print "$iparr[0] <br />";
   print "$iparr[1] <br />" ;
   print "$iparr[2] <br />"  ;
   print "$iparr[3] <br />"  ;

?>
```

This will produce only three strings as we have limited number of strings to be produced.



```
123
456
789.000
```

PHP – Function sql_regcase()

Syntax

```
string sql_regcase (string string)
```

Definition and Usage

The sql_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

If the alphabetical character has both an uppercase and a lowercase format, the bracket will contain both forms; otherwise the original character will be repeated twice.

Return Value

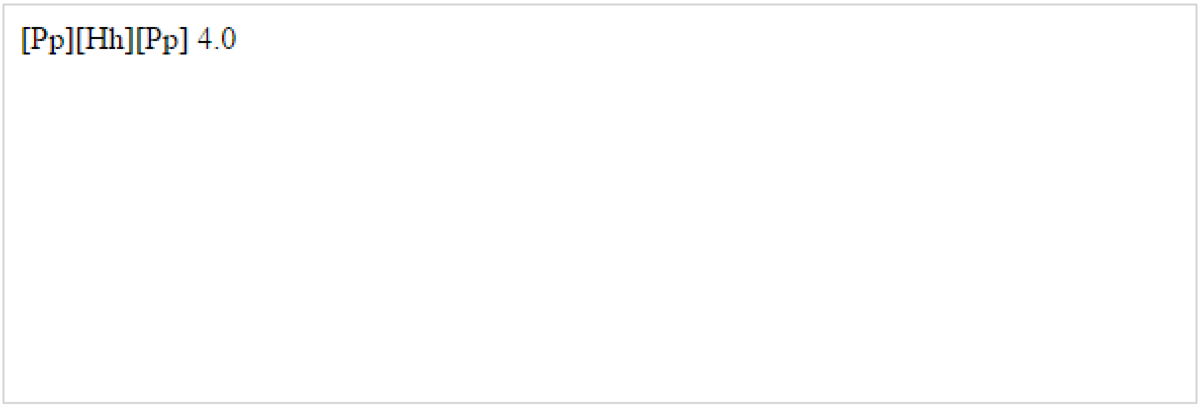
- Returns a string of bracketed expression along with the converted character.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $version = "php 4.0";

   print sql_regcase($version);
?>
```

This will produce the following result –



```
[Pp][Hh][Pp] 4.0
```

PERL Style Regular Expressions

Perl-style regular expressions are similar to their POSIX counterparts. The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions. In fact, you can use any of the quantifiers introduced in the previous POSIX section.

Let us discuss a few concepts being used in PERL regular expressions. After that, we will introduce you with regular expression related functions.

Metacharacters

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for large money sums using the '\d' metacharacter: **/([\d]+)000/**, Here **\d** will search for any string of numerical character.

Following is the list of metacharacters which can be used in PERL Style Regular Expressions.

```
Character         Description
.              a single character
\s             a whitespace character (space, tab, newline)
\S             non-whitespace character
\d             a digit (0-9)
\D             a non-digit
\w             a word character (a-z, A-Z, 0-9, _)
\W             a non-word character
[aeiou]        matches a single character in the given set
[^aeiou]       matches a single character outside the given set
(foo|bar|baz)  matches any of the alternatives specified
```

Modifiers

Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

```
Modifier    Description
i     Makes the match case insensitive
m     Specifies that if the string has newline or carriage
      return characters, the ^ and $ operators will now
      match against a newline boundary, instead of a
      string boundary
o     Evaluates the expression only once
s     Allows use of . to match a newline character
x     Allows you to use white space in the expression for clarity
g     Globally finds all matches
cg    Allows a search to continue even after a global match fails
```

PHP's Regexp PERL Compatible Functions

PHP offers the following functions for searching strings using Perl-compatible regular expressions:

Function	Description
preg_match()	The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.
preg_match_all()	The preg_match_all() function matches all occurrences of pattern in string.
preg_replace()	The preg_replace() function operates just like ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.
preg_split()	The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
preg_grep()	The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
preg_ quote()	Quote regular expression characters

PHP – Function preg_match()

Syntax

```
int preg_match (string pattern, string string [, array pattern_array], [, int
$flags [, int $offset]]]);
```

Definition and Usage

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

If the optional input parameter pattern_array is provided, then pattern_array will contain various sections of the subpatterns contained in the search pattern, if applicable.

If this flag is passed as PREG_OFFSET_CAPTURE, for every occurring match the appendant string offset will also be returned

Normally, the search starts from the beginning of the subject string. The optional parameter offset can be used to specify the alternate place from which to start the search.

Return Value

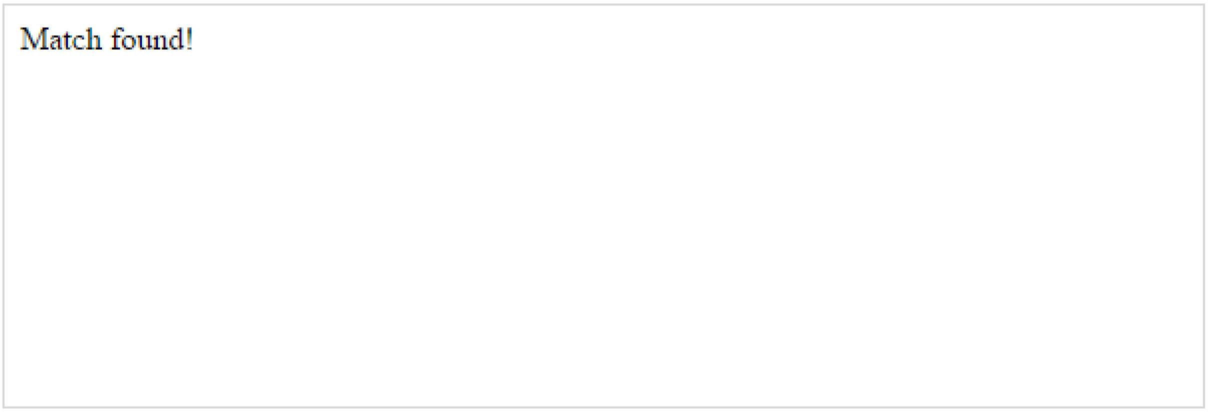
- Returns true if pattern exists, and false otherwise.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $line = "Vi is the greatest word processor ever created!";
   // perform a case-Insensitive search for the word "Vi"

   if (preg_match("/\bVi\b/i", $line, $match)) :
      print "Match found!";
      endif;
?>
```

This will produce the following result –



```
Match found!
```

PHP – Function preg_match_all()

Syntax

```
int preg_match_all (string pattern, string string, array pattern_array [, int
order]);
```

Definition and Usage

The preg_match_all() function matches all occurrences of pattern in string.

It will place these matches in the array pattern_array in the order you specify using the optional input parameter order. There are two possible types of order –

- **PREG_PATTERN_ORDER** –REG_PATTERN_ORDERe matches in the array pattern_array in the od. PREG_PATTERN_ORDER specifies the order in the way that you might think most logical; $pattern_array[0] is an array of all complete pattern matches, $pattern_array[1] is an array of all strings matching the first parenthesized regexp, and so on.

- **PREG_SET_ORDER** –REG_SET_ORDERRDERe matches in the array pattern_array in the od. PREG_PATTERN_ORDER specifies the order in the way that you migparenthesized regexp, $pattern_array[1] will contain elements matched by the second parenthesized regexp, and so on.

Return Value

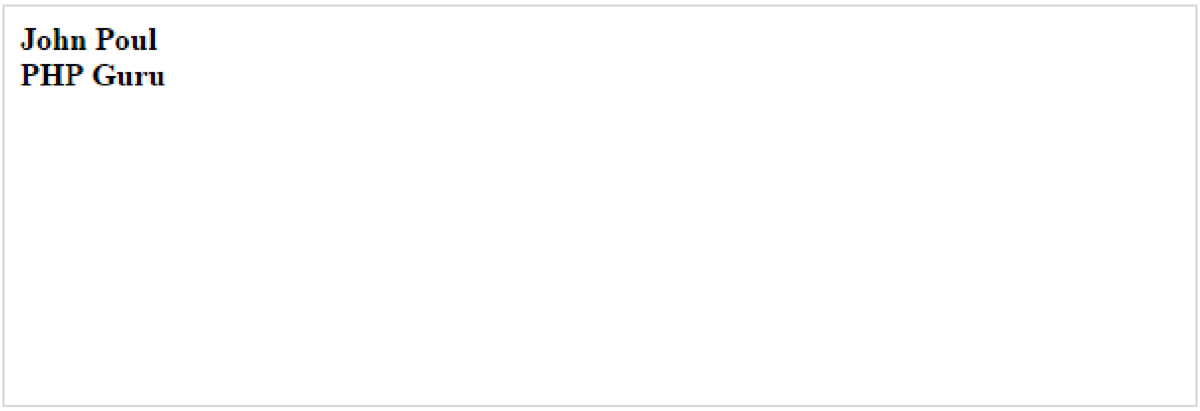
- Returns the number of matchings.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $userinfo = "Name: <b>John Poul</b> <br> Title: <b>PHP Guru</b>";
   preg_match_all ("/<b>(.*)<\/b>/U", $userinfo, $pat_array);

   print $pat_array[0][0]." <br> ".$pat_array[0][1]."\n";
?>
```

This will produce the following result –



```
John Poul
PHP Guru
```

PHP – Function preg_replace()

Syntax

```
mixed preg_replace (mixed pattern, mixed replacement, mixed string [, int limit
[, int &$count]] );
```

Definition and Usage

The preg_replace() function operates just like POSIX function ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.

The optional input parameter limit specifies how many matches should take place.

If the optional parameter $count is passed, then this variable will be filled with the number of replacements done.

Return Value

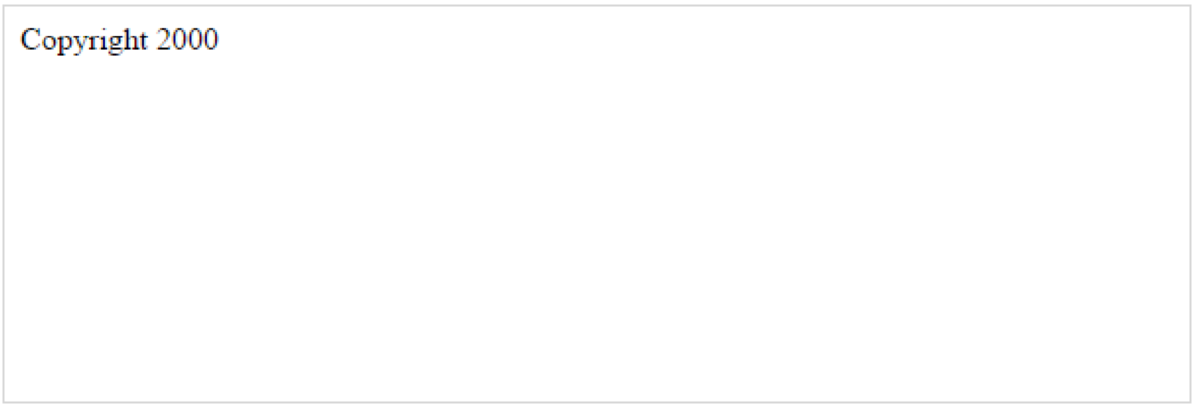
- After the replacement has occurred, the modified string will be returned.
- If no matches are found, the string will remain unchanged.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $copy_date = "Copyright 1999";
   $copy_date = preg_replace("([0-9]+)", "2000", $copy_date);

   print $copy_date;
?>
```

This will produce the following result –



```
Copyright 2000
```

PHP – Function preg_split()

Syntax

```
array preg_split (string pattern, string string [, int limit [, int flags]]);
```

Definition and Usage

The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.

If the optional input parameter limit is specified, then only limit number of substrings are returned.

flags can be any combination of the following types –

- **PREG_SPLIT_NO_EMPTY** –REG_SPLIT_NO_EMPTYmbination of the following fied, then only limit number of s
- **PREG_SPLIT_DELIM_CAPTURE** –REG_SPLIT_DELIM_CAPTUREtion of the following fied, then only limit number of substrings are returned.as input p
- **PREG_SPLIT_OFFSET_CAPTURE** –REG_SPLIT_Oag is set, for every occurring match the appendant string offset will also be returned.

Return Value

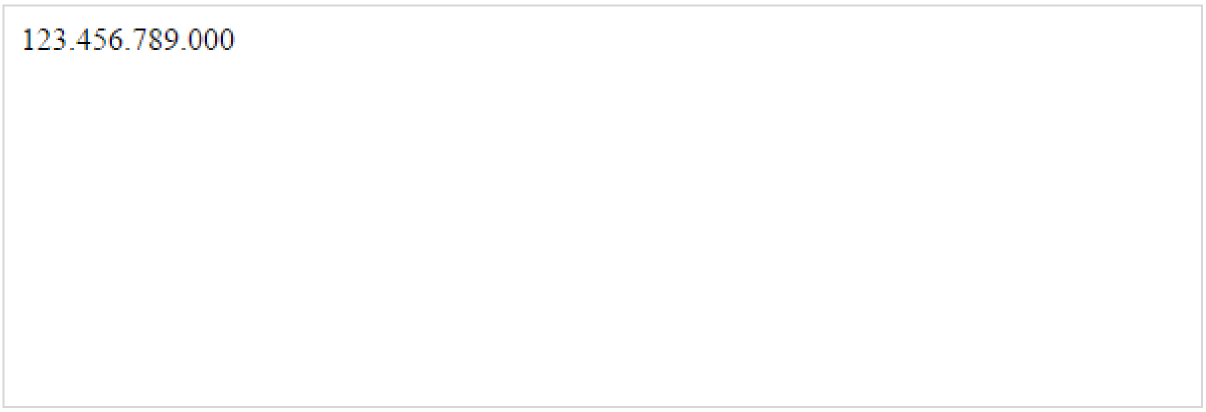
- Returns an array of strings after splitting up a string.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $ip = "123.456.789.000"; // some IP address
   $iparr = split ("/\./", $ip);

   print "$iparr[0] <br />";
   print "$iparr[1] <br />" ;
   print "$iparr[2] <br />"  ;
   print "$iparr[3] <br />"  ;
?>
```

This will produce the following result –



```
123.456.789.000
```

PHP – Function preg_grep()

Syntax

```
array preg_grep ( string $pattern, array $input [, int $flags] );
```

Definition and Usage

Returns the array consisting of the elements of the input array that match the given pattern.

If flag is set to PREG_GREP_INVERT, this function returns the elements of the input array that do not match the given pattern.

Return Value

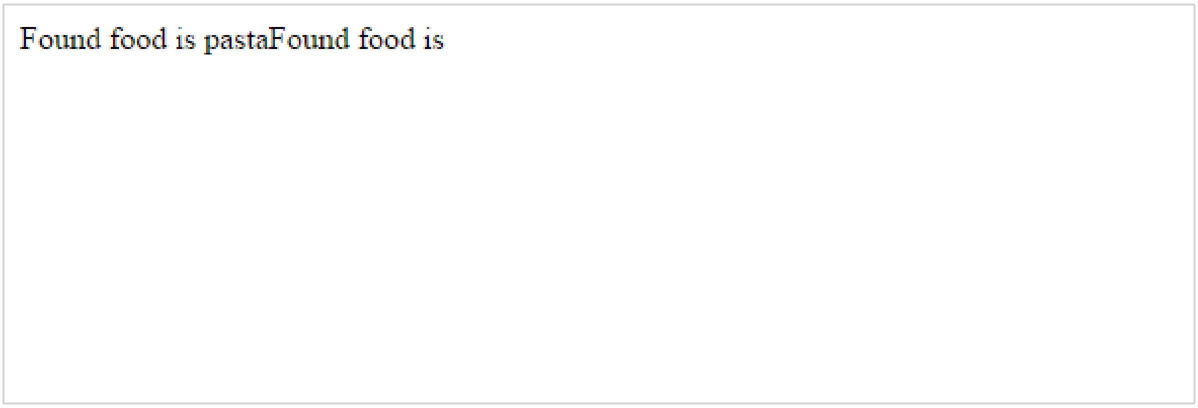
- Returns an array indexed using the keys from the input array.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $foods = array("pasta", "steak", "fish", "potatoes");

   // find elements beginning with "p", followed by one or more letters.
   $p_foods = preg_grep("/p(\w+)/", $foods);

   print "Found food is " . $p_foods[0];
   print "Found food is " . $p_foods[1];
?>
```

This will produce the following result –



```
Found food is pastaFound food is
```

PHP – Function preg_quote()

Syntax

```
string preg_quote ( string $str [, string $delimiter] );
```

Definition and Usage

preg_quote() takes str and puts a backslash in front of every character that is part of the regular expression syntax.

Return Value

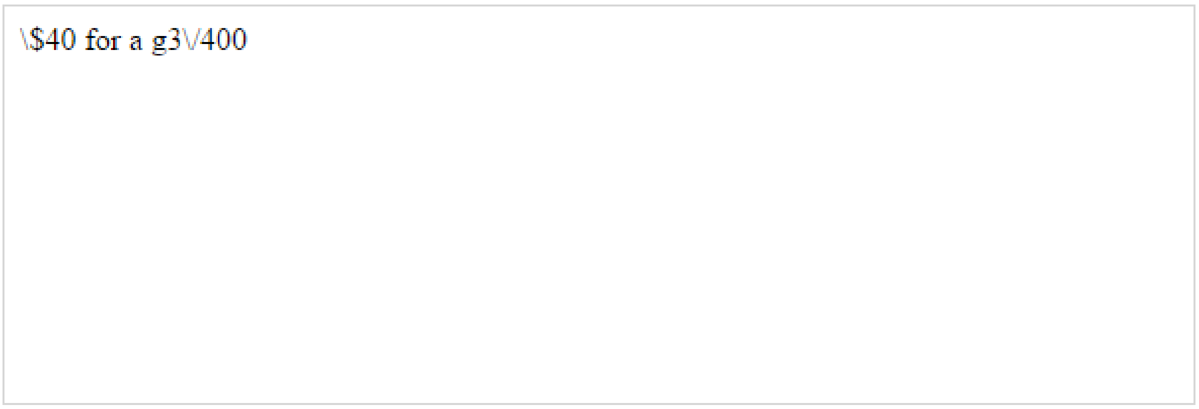
- Returns the quoted string.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
   $keywords = '$40 for a g3/400';
   $keywords = preg_quote($keywords, '/');

   echo $keywords;
?>
```

This will produce the following result –



```
\$40 for a g3\/400
```

23. PHP – Error and Exception Handling

Error handling is the process of catching errors raised by your program and then taking appropriate action. If you would handle errors properly, then it may lead to many unforeseen consequences. It is very simple in PHP to handle errors.

Using die() function

While writing your PHP program you should check all possible error condition before going ahead and take appropriate action when required.

Try the following example without having **/tmp/test.xt** file and with this file.

```
<?php
if(!file_exists("/tmp/test.txt"))
 {
 die("File not found");
 }
else
 {
 $file=fopen("/tmp/test.txt","r");
 print "Opend file sucessfully";
 }
 // Test of the code here.
?>
```

You can thus write an efficient code. Using the above technique, you can stop your program whenever it errors out and display more meaningful and user-friendly message.

Defining Custom Error Handling Function

You can write your own function to handling any error. PHP provides you a framework to define error-handling function.
This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

Syntax

```
error_function(error_level,error_message, error_file,error_line,error_context);
```

Parameter	Description
error_level	Required - Specifies the error report level for the user-defined error. Must be a value number.
error_message	Required - Specifies the error message for the user-defined error
error_file	Optional - Specifies the filename in which the error occurred
error_line	Optional - Specifies the line number in which the error occurred
error_context	Optional - Specifies an array containing every variable and their values in use when the error occurred

Possible Error levels

These error report levels are the different types of error the user-defined error handler can be used for. These values cab used in combination using **|** operator

Value	Constant	Description
1	E_ERROR	Fatal run-time errors. Execution of the script is halted
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
4	E_PARSE	Compile-time parse errors. Parse errors should only be generated by the parser.
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
16	E_CORE_ERROR	Fatal errors that occur during PHP's initial startup.
32	E_CORE_WARNING	Non-fatal run-time errors. This occurs during PHP's initial startup.
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()

2048	E_STRICT	Run-time notices. Enable to have PHP suggest changes to your code which will ensure the best interoperability and forward compatibility of your code.
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())
8191	E_ALL	All errors and warnings, except level E_STRICT (E_STRICT will be part of E_ALL as of PHP 6.0)

All the above error level can be set using the following PHP built-in library function where level cab be any of the value defined in above table.

```
int error_reporting ( [int $level] )
```

Here is how you can create an error handling function:

```
<?php
function handleError($errno, $errstr,$error_file,$error_line)
{
 echo "<b>Error:</b> [$errno] $errstr - $error_file:$error_line";

 echo "<br />";

 echo "Terminating PHP Script";

 die();

}

?>
```

Once you define your custom error handler, you need to set it using PHP built-in library **set_error_handler** function. Now let’s examine our example by calling a function which does not exist.

```
<?php

error_reporting( E_ERROR );

function handleError($errno, $errstr,$error_file,$error_line)

{

 echo "<b>Error:</b> [$errno] $errstr - $error_file:$error_line";

 echo "<br />";

 echo "Terminating PHP Script";

 die();

}

//set error handler

set_error_handler("handleError");


//trigger error
```

```
myFunction();
?>
```

Exceptions Handling

PHP 5 has an exception model similar to that of other programming languages. Exceptions are important and provides a better control over error handling.

Let’s now explain the new keyword related to exceptions.

- **Try -** A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown".
- **Throw -** This is how you trigger an exception. Each "throw" must have at least one "catch".
- **Catch -** - A "catch" block retrieves an exception and creates an object containing the exception information.

When an exception is thrown, the code following the statement will not be executed, and PHP will attempt to find the first matching catch block. If an exception is not caught, a PHP Fatal Error will be issued with an "Uncaught Exception ...

- An exception can be thrown, and caught ("catched") within PHP. Code may be surrounded in a try block.
- Each try must have at least one corresponding catch block. Multiple catch blocks can be used to catch different classes of exceptions.
- Exceptions can be thrown (or re-thrown) within a catch block.

Example

Copy and paste the following piece of code into a file and verify the result.

```
<?php
try {
    $error = 'Always throw this error';
    throw new Exception($error);

    // Code following an exception is not executed.
    echo 'Never executed';

} catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(), "\n";
}
```

```
// Continue execution
echo 'Hello World';
?>
```

In the above example, $e->getMessage function is used to get error message. The following functions can be used from **Exception** class.

- **getMessage()-** message of exception
- **getCode() -** code of exception
- **getFile() -** source filename
- **getLine() -** source line
- **getTrace() -** n array of the backtrace()
- **getTraceAsString() -** formated string of trace

Creating Custom Exception Handler

You can define your own custom exception handler. Use the following function to set a user-defined exception handler function.

```
string set_exception_handler ( callback $exception_handler )
```

Here **exception_handler** is the name of the function to be called when an uncaught exception occurs. This function must be defined before calling set_exception_handler().

Example

```
<?php
function exception_handler($exception) {
  echo "Uncaught exception: " , $exception->getMessage(), "\n";
}

set_exception_handler('exception_handler');

throw new Exception('Uncaught Exception');

echo "Not Executed\n";
?>
```

Check the complete set of error handling functions at PHP Error Handling Functions.

24. PHP – Bugs Debugging

Programs rarely work correctly the first time. Many things can go wrong in your program that cause the PHP interpreter to generate an error message. You have a choice about where those error messages go. The messages can be sent along with other program output to the web browser. They can also be included in the web server error log.

To make error messages display in the browser, set the **display_errors** configuration directive to **On**. To send errors to the web server error log, set **log_errors** to On. You can set them both to On if you want error messages in both places.

PHP defines some constants you can use to set the value of **error_reporting** such that only errors of certain types get reported: E_ALL (for all errors except strict notices), E_PARSE (parse errors), E_ERROR (fatal errors), E_WARNING (warnings), E_NOTICE (notices), and E_STRICT (strict notices).

While writing your PHP program, it is a good idea to use PHP-aware editors like **BBEdit** or **Emacs**. One of the special features of these editors is syntax highlighting. It changes the color of different parts of your program based on what those parts are. For example, strings are pink, keywords such as if and while are blue, comments are grey, and variables are black.

Another feature is quote and bracket matching, which helps to make sure that your quotes and brackets are balanced. When you type a closing delimiter such as }, the editor highlights the opening { that it matches.

You need to verify the following points while debugging your program.

- **Missing Semicolons -** Every PHP statement ends with a semicolon (;). PHP doesn't stop reading a statement until it reaches a semicolon. If you leave out the semicolon at the end of a line, PHP continues reading the statement on the following line.

- **Not Enough Equal Signs -** When you ask whether two values are equal in a comparison statement, you need two equal signs (==). Using one equal sign is a common mistake.

- **Misspelled Variable Names -** If you misspelled a variable, then PHP understands it as a new variable. Remember: To PHP, $test is not the same variable as $Test.

- **Missing Dollar Signs -** A missing dollar sign in a variable name is really hard to see, but at least it usually results in an error message so that you know where to look for the problem.

- **Troubling Quotes -** You can have too many, too few, or the wrong kind of quotes. So check for a balanced number of quotes.

- **Missing Parentheses and curly brackets -** They should always be in pairs.

- **Array Index -** All the arrays should start from zero instead of 1.

Moreover, handle all the errors properly and direct all trace messages into system log file so that if any problem occurs, then it will be logged into system log file and you will be able to debug that problem.

25. PHP – Date and Time

Dates are so much part of everyday life that it becomes easy to work with them without thinking. PHP also provides powerful tools for date arithmetic that make manipulating dates easy.

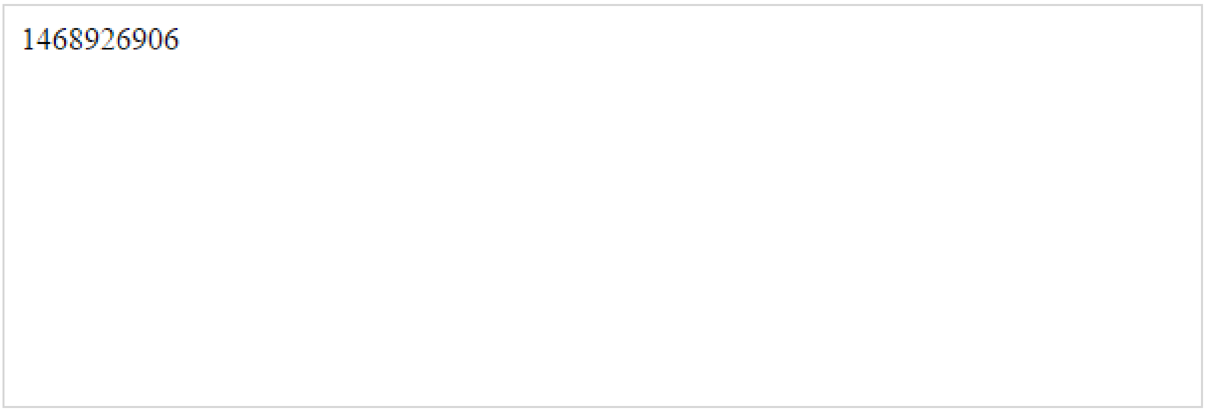
Getting the Time Stamp with time()

PHP's **time()** function gives you all the information that you need about the current date and time. It requires no arguments but returns an integer.

The integer returned by time() represents the number of seconds elapsed since midnight GMT on January 1, 1970. This moment is known as the UNIX epoch, and the number of seconds that have elapsed since then is referred to as a time stamp.

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

It will produce the following result:



```
1468926906
```

This is something difficult to understand. But PHP offers excellent tools to convert a time stamp into a form that humans are comfortable with.

Converting a Time Stamp with getdate()

The function **getdate()** optionally accepts a timestamp and returns an associative array containing information about the date. If you omit the time stamp, it works with the current time stamp as returned by time().

The following table lists the elements contained in the array returned by getdate().

Key	Description	Example
seconds	Seconds past the minutes (0-59)	20
minutes	Minutes past the hour (0 - 59)	29
hours	Hours of the day (0 - 23)	22
mday	Day of the month (1 - 31)	11
wday	Day of the week (0 - 6)	4
mon	Month of the year (1 - 12)	7
year	Year (4 digits)	1997
yday	Day of year (0 - 365)	19
weekday	Day of the week	Thursday
month	Month of the year	January
0	Timestamp	948370048

Now you have complete control over date and time. You can format this date and time in whatever format you want.

Example

Try out the following example.

```
<?php
$date_array = getdate();
foreach ( $date_array as $key => $val )
{
   print "$key = $val<br />";
}
$formated_date  = "Today's date: ";
$formated_date .= $date_array[mday] . "/";
$formated_date .= $date_array[mon] . "/";
$formated_date .= $date_array[year];

print $formated_date;
?>
```

It will produce the following result:

```
seconds = 6
minutes = 15
hours = 11
mday = 19
wday = 2
mon = 7
year = 2016
yday = 200
weekday = Tuesday
month = July
0 = 1468926906
Today's date: 19/7/2016
```

Converting a Time Stamp with date()

The **date()** function returns a formatted string representing a date. You can exercise an enormous amount of control over the format that date() returns with a string argument that you must pass to it.

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

The date() optionally accepts a time stamp if omitted, then current date and time will be used. Any other data you include in the format string passed to date() will be included in the return value.

The following table lists the codes that a format string can contain:

Format	Description	Example
a	'am' or 'pm' lowercase	pm
A	'AM' or 'PM' uppercase	PM
d	Day of month, a number with leading zeroes	20
D	Day of week (three letters)	Thu
F	Month name	January
h	Hour (12-hour format - leading zeroes)	12
H	Hour (24-hour format - leading zeroes)	22
g	Hour (12-hour format - no leading zeroes)	12
G	Hour (24-hour format - no leading zeroes)	22
i	Minutes (0 - 59)	23

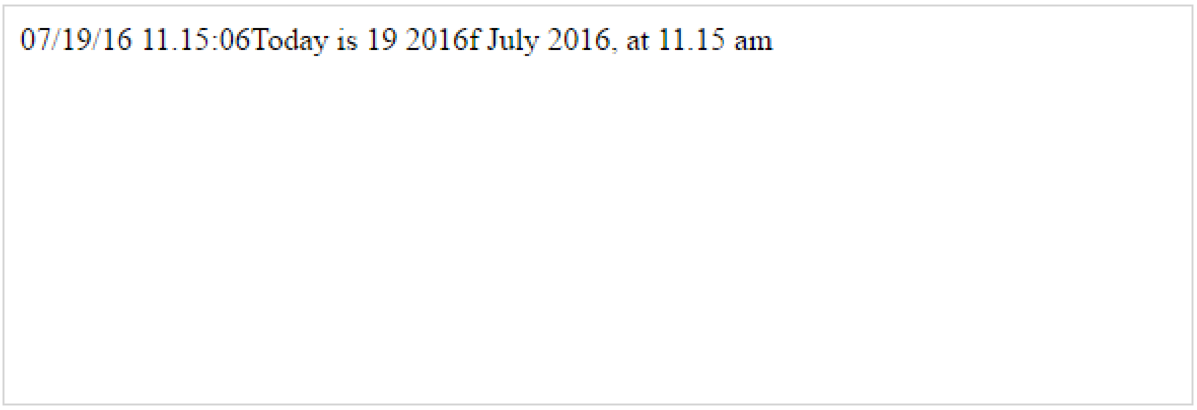
j	Day of the month (no leading zeroes	20
l (Lower 'L')	Day of the week	Thursday
L	Leap year ('1' for yes, '0' for no)	1
m	Month of year (number - leading zeroes)	1
M	Month of year (three letters)	Jan
r	The RFC 2822 formatted date	Thu, 21 Dec 2000 16:01:07 +0200
n	Month of year (number - no leading zeroes)	2
s	Seconds of hour	20
U	Time stamp	948372444
y	Year (two digits)	06
Y	Year (four digits)	2006
z	Day of year (0 - 365)	206
Z	Offset in seconds from GMT	+5

Example

Try out the following example.

```
<?php
print date("m/d/y G.i:s<br>", time());
print "Today is ";
print date("j of F Y, \a\\t g.i a", time());
?>
```

It will produce following result:



```
07/19/16 11.15:06Today is 19 2016f July 2016, at 11.15 am
```

Hope you have a good understanding of how to format date and time according to your requirement. For your reference a complete list of all the date and time functions is given in PHP Date & Time Functions.

26. PHP – PHP and MySQL

PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.

What you should already have?

- You have gone through MySQL tutorial to understand MySQL Basics.
- Downloaded and installed a latest version of MySQL.
- Created database user **guest** with password **guest123**.
- If you have not created a database, then you would need root user and its password to create a database.

We have divided this chapter in the following sections:

- **Connecting to MySQL database** - Learn how to use PHP to open and close a MySQL database connection.
- **Create MySQL Database Using PHP** - This part explains how to create MySQL database and tables using PHP.
- **Delete MySQL Database Using PHP** - This part explains how to delete MySQL database and tables using PHP.
- **Insert Data To MySQL Database** - Once you have created your database and tables, then you would like to insert your data into created tables. This session will take you through real example on data insert.
- **Retrieving Data From MySQL Database** - Learn how to fetch records from MySQL database using PHP.
- **Using Paging through PHP** - This one explains how to show your query result into multiple pages and how to create the navigation link.
- **Updating Data Into MySQL Database** - This part explains how to update existing records into MySQL database using PHP.
- **Deleting Data From MySQL Database** - This part explains how to delete or purge existing records from MySQL database using PHP.
- **Using PHP To Backup MySQL Database** - Learn different ways to take backup of your MySQL database for safety purpose.

Connecting to MySQL Database

Opening a Database Connection

PHP provides **mysql_connect** function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

Syntax

```
connection mysql_connect(server,user,passwd,new_link,client_flag);
```

Sr.No	Parameter & Description
1	**server** Optional – The host name running database server. If not specified, then default value is **localhost:3306**.
2	**user** Optional – The username accessing the database. If not specified, then default is the name of the user that owns the server process.
3	**passwd** Optional – The password of the user accessing the database. If not specified then default is an empty password.
4	**new_link** Optional – If a second call is made to mysql_connect() with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned.
5	**client_flags** Optional – A combination of the following constants – • **MYSQL_CLIENT_SSL** – Use SSL encryption • **MYSQL_CLIENT_COMPRESS** – Use compression protocol

	• **MYSQL_CLIENT_IGNORE_SPACE** – Allow space after function names • **MYSQL_CLIENT_INTERACTIVE** – Allow interactive timeout seconds of inactivity before closing the connection

NOTE – You can specify server, user, passwd in **php.ini** file instead of using them again and again in your every PHP scripts. Check php.ini fileconfiguration.

Closing Database Connection

Its simplest function **mysql_close** PHP provides to close a database connection. This function takes connection resource returned by mysql_connect function. It returns TRUE on success or FALSE on failure.

Syntax

```
bool mysql_close ( resource $link_identifier );
```

If a resource is not specified, then the last opened database is closed.

Example

Try the following example to open and close a database connection –

```
<?php

   $dbhost = 'localhost:3036';
   $dbuser = 'guest';
   $dbpass = 'guest123';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   echo 'Connected successfully';
   mysql_close($conn);
?>
```

Create MySQL Database Using PHP

Creating a Database

To create and delete a database, you should have admin privilege. It's very easy to create a new MySQL database. PHP uses **mysql_query** function to create a MySQL database. This function takes two parameters and returns TRUE on success or FALSE on failure.

Syntax

```
bool mysql_query( sql, connection );
```

Sr.No	Parameter & Description
1	**sql** Required - SQL query to create a database
2	**connection** Optional - if not specified, then the last opened connection by mysql_connect will be used.

Example

Try the following example to create a database –

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   echo 'Connected successfully';

   $sql = 'CREATE Database test_db';
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not create database: ' . mysql_error());
   }


   echo "Database test_db created successfully\n";
   mysql_close($conn);
?>
```

Selecting a Database

Once you establish a connection with a database server, then it is required to select a particular database with which all your tables are associated.

This is required because there may be multiple databases residing on a single server and you can do work with a single database at a time.

PHP provides function **mysql_select_db** to select a database. It returns TRUE on success or FALSE on failure.

Syntax

```
bool mysql_select_db( db_name, connection );
```

Sr.No	Parameter & Description
1	**db_name** Required - Database name to be selected
2	**connection** Optional - if not specified, then the last opened connection by mysql_connect will be used.

Example

Here is the example showing you how to select a database.

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'guest';
   $dbpass = 'guest123';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   echo 'Connected successfully';

   mysql_select_db( 'test_db' );
   mysql_close($conn);

?>
```

Creating Database Tables

To create tables in the new database, you need to do the same thing as creating the database. First create the SQL query to create the tables, then execute the query using mysql_query() function.

Example

Try the following example to create a table –

```
<?php

   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   echo 'Connected successfully';

   $sql = 'CREATE TABLE employee( '.
      'emp_id INT NOT NULL AUTO_INCREMENT, '.
      'emp_name VARCHAR(20) NOT NULL, '.
      'emp_address  VARCHAR(20) NOT NULL, '.
      'emp_salary   INT NOT NULL, '.
      'join_date    timestamp(14) NOT NULL, '.
      'primary key ( emp_id ))';
   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not create table: ' . mysql_error());
   }

   echo "Table employee created successfully\n";

   mysql_close($conn);
?>
```

In case you need to create many tables, then it's better to create a text file first and put all the SQL commands in that text file and then load that file into $sql variable and execute those commands.

Consider the following content in sql_query.txt file

```
CREATE TABLE employee(
   emp_id INT NOT NULL AUTO_INCREMENT,
   emp_name VARCHAR(20) NOT NULL,
   emp_address  VARCHAR(20) NOT NULL,
   emp_salary   INT NOT NULL,
   join_date    timestamp(14) NOT NULL,
   primary key ( emp_id ));
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $query_file = 'sql_query.txt';

   $fp = fopen($query_file, 'r');
   $sql = fread($fp, filesize($query_file));
   fclose($fp);
   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not create table: ' . mysql_error());
   }

   echo "Table employee created successfully\n";
   mysql_close($conn);
?>
```

Delete MySQL Database Using PHP

Deleting a Database

If a database is no longer required, then it can be deleted forever. You can use pass an SQL command to **mysql_query** to delete a database.

Example

Try the following example to drop a database.

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $sql = 'DROP DATABASE test_db';
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not delete database db_test: ' . mysql_error());
   }

   echo "Database deleted successfully\n";
   mysql_close($conn);
?>
```

WARNING – It's very dangerous to delete a database and any table. So before deleting any table or database you should make sure you are doing everything intentionally.

Deleting a Table

It's again a matter of issuing one SQL command through **mysql_query** function to delete any database table. But be very careful while using this command because by doing so you can delete some important information you have in your table.

Example

Try the following example to drop a table –

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $sql = 'DROP TABLE employee';
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not delete table employee: ' . mysql_error());
   }

   echo "Table deleted successfully\n";

   mysql_close($conn);
?>
```

Insert Data to MySQL Database

Data can be entered into MySQL tables by executing SQL INSERT statement through PHP function **mysql_query**. Below a simple example to insert a record into **employee** table.

Example

Try the following example to insert record into employee table.

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $sql = 'INSERT INTO employee '.
      '(emp_name,emp_address, emp_salary, join_date) '.
      'VALUES ( "guest", "XYZ", 2000, NOW() )';

   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not enter data: ' . mysql_error());
   }

   echo "Entered data successfully\n";

   mysql_close($conn);
?>
```

In real application, all the values will be taken using HTML form and then those values will be captured using PHP script and finally they will be inserted into MySQL tables.

While doing data insert its best practice to use function **get_magic_quotes_gpc()** to check if current configuration for magic quote is set or not. If this function returns false, then use function **addslashes()** to add slashes before quotes.

Example

Try this example by putting this code into add_employee.php, this will take input using HTML Form and then it will create records into database.

```
<html>
```

```
   <head>
      <title>Add New Record in MySQL Database</title>
   </head>

   <body>
      <?php
         if(isset($_POST['add'])) {
            $dbhost = 'localhost:3036';
            $dbuser = 'root';
            $dbpass = 'rootpassword';
            $conn = mysql_connect($dbhost, $dbuser, $dbpass);

            if(! $conn ) {
               die('Could not connect: ' . mysql_error());
            }

            if(! get_magic_quotes_gpc() ) {
               $emp_name = addslashes ($_POST['emp_name']);
               $emp_address = addslashes ($_POST['emp_address']);
            }else {
               $emp_name = $_POST['emp_name'];
               $emp_address = $_POST['emp_address'];
            }

            $emp_salary = $_POST['emp_salary'];

            $sql = "INSERT INTO employee ". "(emp_name,emp_address, emp_salary,
               join_date) ". "VALUES('$emp_name','$emp_address',$emp_salary, NOW())";

            mysql_select_db('test_db');
            $retval = mysql_query( $sql, $conn );

            if(! $retval ) {
               die('Could not enter data: ' . mysql_error());
            }

            echo "Entered data successfully\n";

            mysql_close($conn);
         }else {
```

```
         ?>

           <form method = "post" action = "<?php $_PHP_SELF ?>">
              <table width = "400" border = "0" cellspacing = "1"
                 cellpadding = "2">

                 <tr>
                    <td width = "100">Employee Name</td>
                    <td><input name = "emp_name" type = "text"
                       id = "emp_name"></td>
                 </tr>

                 <tr>
                    <td width = "100">Employee Address</td>
                    <td><input name = "emp_address" type = "text"
                       id = "emp_address"></td>
                 </tr>

                 <tr>
                    <td width = "100">Employee Salary</td>
                    <td><input name = "emp_salary" type = "text"
                       id = "emp_salary"></td>
                 </tr>

                 <tr>
                    <td width = "100"> </td>
                    <td> </td>
                 </tr>

                 <tr>
                    <td width = "100"> </td>
                    <td>
                       <input name = "add" type = "submit" id = "add"
                          value = "Add Employee">
                    </td>
                 </tr>

              </table>
           </form>
```

```
            <?php
         }
      ?>

   </body>
</html>
```

Retrieving Data from MySQL Database

Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function mysql_query. You have several options to fetch data from MySQL.

The most frequently used option is to use function **mysql_fetch_array()**. This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

Following is a simple example to fetch records from **employee** table.

Example

Try the following example to display all the records from employee table.

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';

   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not get data: ' . mysql_error());
   }

   while($row = mysql_fetch_array($retval, MYSQL_ASSOC)) {
      echo "EMP ID :{$row['emp_id']}  <br> ".
         "EMP NAME : {$row['emp_name']} <br> ".
         "EMP SALARY : {$row['emp_salary']} <br> ".
         "--------------------------------<br>";
   }

   echo "Fetched data successfully\n";

   mysql_close($conn);
?>
```

The content of the rows are assigned to the variable $row and the values in row are then printed.

NOTE – Always remember to put curly brackets when you want to insert an array value directly into a string.

In the above example, the constant **MYSQL_ASSOC** is used as the second argument to mysql_fetch_array(), so that it returns the row as an associative array. With an associative array, you can access the field by using their name instead of using the index.

PHP provides another function called **mysql_fetch_assoc()** which also returns the row as an associative array.

Example

Try the following example to display all the records from employee table using mysql_fetch_assoc() function.

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';

   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );
   if(! $retval ) {
      die('Could not get data: ' . mysql_error());
   }

   while($row = mysql_fetch_assoc($retval)) {
      echo "EMP ID :{$row['emp_id']}  <br> ".
         "EMP NAME : {$row['emp_name']} <br> ".
         "EMP SALARY : {$row['emp_salary']} <br> ".
         "--------------------------------<br>";
   }

   echo "Fetched data successfully\n";

   mysql_close($conn);
?>
```

You can also use the constant **MYSQL_NUM**, as the second argument to mysql_fetch_array(). This will cause the function to return an array with numeric index.

Example

Try the following example to display all the records from employee table using MYSQL_NUM argument.

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';

   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );
   if(! $retval ) {
      die('Could not get data: ' . mysql_error());
   }

   while($row = mysql_fetch_array($retval, MYSQL_NUM)) {
      echo "EMP ID :{$row[0]}  <br> ".
         "EMP NAME : {$row[1]} <br> ".
         "EMP SALARY : {$row[2]} <br> ".
         "--------------------------------<br>";
   }

   echo "Fetched data successfully\n";

   mysql_close($conn);
?>
```

All the above three examples will produce the same result.

Releasing Memory

It is a good practice to release cursor memory at the end of each SELECT statement. This can be done by using PHP function **mysql_free_result()**. Below is the example to show how it has to be used.

Example

Try the following example.

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';

   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not get data: ' . mysql_error());
   }

   while($row = mysql_fetch_array($retval, MYSQL_NUM)) {
      echo "EMP ID :{$row[0]}  <br> ".
         "EMP NAME : {$row[1]} <br> ".
         "EMP SALARY : {$row[2]} <br> ".
         "--------------------------------<br>";
   }

   mysql_free_result($retval);
   echo "Fetched data successfully\n";

   mysql_close($conn);
?>
```

While fetching data, you can write as complex SQL as you like. The procedure will remain the same as mentioned above.

Using Paging through PHP

It's always possible that your SQL SELECT statement query may result into thousands of records. But it is not good idea to display all the results on one page. So we can divide this result into many pages as per requirement.

Paging means showing your query result in multiple pages instead of just put them all in one long page.

MySQL helps to generate paging by using **LIMIT** clause which will take two arguments. First argument as OFFSET and second argument how many records should be returned from the database.

Following is a simple example to fetch records using **LIMIT** clause to generate paging.

Example

Try the following example to display 10 records per page.

```
<html>

   <head>
      <title>Paging Using PHP</title>
   </head>

   <body>
      <?php
         $dbhost = 'localhost:3036';
         $dbuser = 'root';
         $dbpass = 'rootpassword';

         $rec_limit = 10;
         $conn = mysql_connect($dbhost, $dbuser, $dbpass);

         if(! $conn ) {
            die('Could not connect: ' . mysql_error());
         }
         mysql_select_db('test_db');

         /* Get total number of records */
         $sql = "SELECT count(emp_id) FROM employee ";
         $retval = mysql_query( $sql, $conn );

         if(! $retval ) {
            die('Could not get data: ' . mysql_error());
         }
         $row = mysql_fetch_array($retval, MYSQL_NUM );
         $rec_count = $row[0];

         if( isset($_GET{'page'} ) ) {
            $page = $_GET{'page'} + 1;
            $offset = $rec_limit * $page ;
         }else {
            $page = 0;
            $offset = 0;
         }

         $left_rec = $rec_count - ($page * $rec_limit);
         $sql = "SELECT emp_id, emp_name, emp_salary ".
            "FROM employee ".
            "LIMIT $offset, $rec_limit";
         $retval = mysql_query( $sql, $conn );

         if(! $retval ) {
            die('Could not get data: ' . mysql_error());
         }

         while($row = mysql_fetch_array($retval, MYSQL_ASSOC)) {
            echo "EMP ID :{$row['emp_id']}  <br> ".
               "EMP NAME : {$row['emp_name']} <br> ".
               EMP SALARY : {$row['emp_salary']} <br> ".
               "--------------------------------<br>";
         }

         if( $page > 0 ) {
            $last = $page - 2;
            echo "<a href = \"$_PHP_SELF?page = $last\">Last 10 Records</a> |";
            echo "<a href = \"$_PHP_SELF?page = $page\">Next 10 Records</a>";
         }else if( $page == 0 ) {
            echo "<a href = \"$_PHP_SELF?page = $page\">Next 10 Records</a>";
         }else if( $left_rec < $rec_limit ) {
            $last = $page - 2;
            echo "<a href = \"$_PHP_SELF?page = $last\">Last 10 Records</a>";
         }

         mysql_close($conn);
      ?>

   </body>
</html>
```

Updating Data into MySQL Database

Data can be updated into MySQL tables by executing SQL UPDATE statement through PHP function **mysql_query**.

Below is a simple example to update records into **employee** table. To update a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in employee table.

Example

Try the following example to understand update operation. You need to provide an employee ID to update an employee salary.

```
<html>

   <head>
      <title>Update a Record in MySQL Database</title>
   </head>

   <body>
      <?php
         if(isset($_POST['update'])) {
            $dbhost = 'localhost:3036';
            $dbuser = 'root';
            $dbpass = 'rootpassword';

            $conn = mysql_connect($dbhost, $dbuser, $dbpass);

            if(! $conn ) {
               die('Could not connect: ' . mysql_error());
            }

            $emp_id = $_POST['emp_id'];
            $emp_salary = $_POST['emp_salary'];

            $sql = "UPDATE employee ". "SET emp_salary = $emp_salary ".
               "WHERE emp_id = $emp_id" ;
            mysql_select_db('test_db');
            $retval = mysql_query( $sql, $conn );

            if(! $retval ) {
               die('Could not update data: ' . mysql_error());
            }
            echo "Updated data successfully\n";

            mysql_close($conn);
         }else {
            ?>
```

```
                <form method = "post" action = "<?php $_PHP_SELF ?>">
                   <table width = "400" border =" 0" cellspacing = "1" 
                      cellpadding = "2">
                   
                      <tr>
                         <td width = "100">Employee ID</td>
                         <td><input name = "emp_id" type = "text" 
                            id = "emp_id"></td>
                      </tr>
                   
                      <tr>
                         <td width = "100">Employee Salary</td>
                         <td><input name = "emp_salary" type = "text" 
                            id = "emp_salary"></td>
                      </tr>
                   
                      <tr>
                         <td width = "100"> </td>
                         <td> </td>
                      </tr>
                   
                      <tr>
                         <td width = "100"> </td>
                         <td>
                            <input name = "update" type = "submit" 
                               id = "update" value = "Update">
                         </td>
                      </tr>
                   
                   </table>
                </form>
             <?php
          }
       ?>
       
    </body>
</html>
```

Deleting Data from MySQL Database

Data can be deleted from MySQL tables by executing SQL DELETE statement through PHP function **mysql_query**.

Following is a simple example to delete records into **employee** table. To delete a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in employee table.

Example

Try the following example to understand delete operation. You need to provide an employee ID to delete an employee record from employee table.

```
<html>


   <head>
      <title>Delete a Record from MySQL Database</title>
   </head>


   <body>
      <?php
         if(isset($_POST['delete'])) {
            $dbhost = 'localhost:3036';
            $dbuser = 'root';
            $dbpass = 'rootpassword';
            $conn = mysql_connect($dbhost, $dbuser, $dbpass);


            if(! $conn ) {
               die('Could not connect: ' . mysql_error());
            }


            $emp_id = $_POST['emp_id'];


            $sql = "DELETE employee ". "WHERE emp_id = $emp_id" ;
            mysql_select_db('test_db');
            $retval = mysql_query( $sql, $conn );


            if(! $retval ) {
               die('Could not delete data: ' . mysql_error());
            }
            echo "Deleted data successfully\n";

            mysql_close($conn);
         }else {
            ?>
               <form method = "post" action = "<?php $_PHP_SELF ?>">
                  <table width = "400" border = "0" cellspacing = "1"
                     cellpadding = "2">

                     <tr>
                        <td width = "100">Employee ID</td>
                        <td><input name = "emp_id" type = "text"
                           id = "emp_id"></td>
                     </tr>

                     <tr>
                        <td width = "100"> </td>
                        <td> </td>
                     </tr>

                     <tr>
                        <td width = "100"> </td>
                        <td>
                           <input name = "delete" type = "submit"
                              id = "delete" value = "Delete">
                        </td>
                     </tr>

                  </table>
               </form>
            <?php
         }
      ?>

   </body>
</html>
```

Using PHP to Backup MySQL Database

It is always a good practice to take a regular backup of your database. There are three ways you can use to take backup of your MySQL database.

- Using SQL Command through PHP.
- Using MySQL binary mysqldump through PHP.
- Using phpMyAdmin user interface.

Using SQL Command through PHP

You can execute SQL SELECT command to take a backup of any table. To take a complete database dump you will need to write separate query for separate table. Each table will be stored into separate text file.

Example

Try the following example of using SELECT INTO OUTFILE query for creating table backup −

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';

   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $table_name = "employee";
   $backup_file  = "/tmp/employee.sql";
   $sql = "SELECT * INTO OUTFILE '$backup_file' FROM $table_name";

   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not take data backup: ' . mysql_error());
   }
   echo "Backedup  data successfully\n";

   mysql_close($conn);
?>
```

There may be instances when you would need to restore data which you have backed up some time ago. To restore the backup, you just need to run LOAD DATA INFILE query like this –

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';

   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $table_name = "employee";
   $backup_file  = "/tmp/employee.sql";
   $sql = "LOAD DATA INFILE '$backup_file' INTO TABLE $table_name";

   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not load data : ' . mysql_error());
   }
   echo "Loaded  data successfully\n";

   mysql_close($conn);
?>
```

Using MySQL binary mysqldump through PHP

MySQL provides one utility **mysqldump** to perform database backup. Using this binary you can take complete database dump in a single command.

Example

Try the following example to take your complete database dump –

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';

   $backup_file = $dbname . date("Y-m-d-H-i-s") . '.gz';
   $command = "mysqldump --opt -h $dbhost -u $dbuser -p $dbpass ". "test_db |
gzip > $backup_file";

   system($command);
?>
```

Using phpMyAdmin user interface

If you have **phpMyAdmin** user interface available, then it is very easy for you to take backup of your database.

To back up your MySQL database using phpMyAdmin click on the "export" link on phpMyAdmin main page. Choose the database you wish to backup, check the appropriate SQL options and enter the name for the backup file.

27. PHP – PHP and AJAX

What is AJAX ?

- AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS and Java Script.

- Conventional web application transmit information to and from the sever using synchronous requests. This means you fill out a form, hit submit, and get directed to a new page with new information from the server.

- With AJAX when submit is pressed, JavaScript will make a request to the server, interpret the results and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.

For complete learning on AJAX, please refer our AJAX Tutorial.

PHP and AJAX Example

To clearly illustrate how easy it is to access information from a database using Ajax and PHP, we are going to build MySQL queries on the fly and display the results on "ajax.html". But before we proceed, let's do a bit of ground work first. Create a table using the following command.

NOTE: We are assuming you have sufficient privilege to perform following MySQL operations

```
CREATE TABLE `ajax_example` (
  `name` varchar(50) NOT NULL,
  `age` int(11) NOT NULL,
  `sex` varchar(1) NOT NULL,
  `wpm` int(11) NOT NULL,
  PRIMARY KEY  (`name`)
)
```

Now dump the following data into this table using the following SQL statements:

```
INSERT INTO `ajax_example` VALUES ('Jerry', 120, 'm', 20);
INSERT INTO `ajax_example` VALUES ('Regis', 75, 'm', 44);
INSERT INTO `ajax_example` VALUES ('Frank', 45, 'm', 87);
INSERT INTO `ajax_example` VALUES ('Jill', 22, 'f', 72);
INSERT INTO `ajax_example` VALUES ('Tracy', 27, 'f', 0);
INSERT INTO `ajax_example` VALUES ('Julie', 35, 'f', 90);
```

Client Side HTML file

Now let's have our client side HTML file which is ajax.html and it will have following code

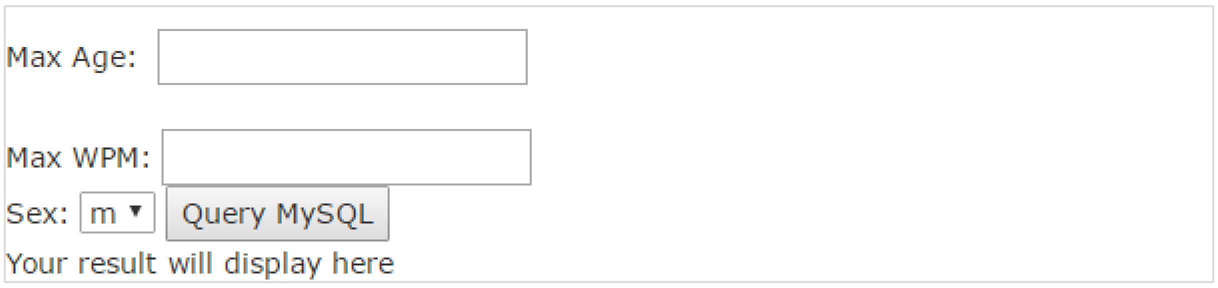
```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
//Browser Support Code
function ajaxFunction(){
 var ajaxRequest;  // The variable that makes Ajax possible!

 try{
   // Opera 8.0+, Firefox, Safari
   ajaxRequest = new XMLHttpRequest();
 }catch (e){
   // Internet Explorer Browsers
   try{
      ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
   }catch (e) {
      try{
         ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
      }catch (e){
         // Something went wrong
         alert("Your browser broke!");
         return false;
      }
   }
 }
 // Create a function that will receive data
 // sent from the server and will update
 // div section in the same page.
 ajaxRequest.onreadystatechange = function(){
   if(ajaxRequest.readyState == 4){
      var ajaxDisplay = document.getElementById('ajaxDiv');
      ajaxDisplay.innerHTML = ajaxRequest.responseText;
   }
 }
 // Now get the value from user and pass it to
 // server script.
 var age = document.getElementById('age').value;
 var wpm = document.getElementById('wpm').value;
 var sex = document.getElementById('sex').value;
 var queryString = "?age=" + age ;
 queryString +=  "&wpm=" + wpm + "&sex=" + sex;
 ajaxRequest.open("GET", "ajax-example.php" +
                              queryString, true);
 ajaxRequest.send(null);
}
//-->
</script>
<form name='myForm'>
Max Age: <input type='text' id='age' /> <br />
Max WPM: <input type='text' id='wpm' />
<br />
Sex: <select id='sex'>
<option value="m">m</option>
<option value="f">f</option>
</select>
<input type='button' onclick='ajaxFunction()'
                              value='Query MySQL'/>
</form>
<div id='ajaxDiv'>Your result will display here</div>
</body>
</html>
```

NOTE: The way of passing variables in the Query is according to HTTP standard and the have formA

```
URL?variable1=value1;&variable2=value2;
```

The above code will produce a screen as given below:

NOTE: This is dummy screen.



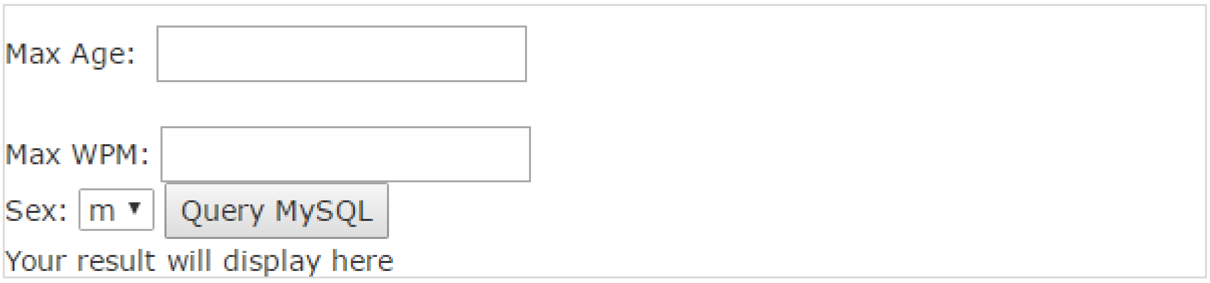
Server Side PHP file

So now your client side script is ready. Now we have to write our server side script which will fetch age, wpm and sex from the database and will send it back to the client. Put the following code into "ajax-example.php" file

```
<?php
$dbhost = "localhost";
$dbuser = "dbusername";
$dbpass = "dbpassword";
$dbname = "dbname";
     //Connect to MySQL Server
mysql_connect($dbhost, $dbuser, $dbpass);
     //Select Database
mysql_select_db($dbname) or die(mysql_error());
     // Retrieve data from Query String
$age = $_GET['age'];
$sex = $_GET['sex'];
$wpm = $_GET['wpm'];
     // Escape User Input to help prevent SQL Injection
$age = mysql_real_escape_string($age);
$sex = mysql_real_escape_string($sex);
$wpm = mysql_real_escape_string($wpm);
     //build query
$query = "SELECT * FROM ajax_example WHERE sex = '$sex'";
if(is_numeric($age))
     $query .= " AND age <= $age";
if(is_numeric($wpm))
     $query .= " AND wpm <= $wpm";
     //Execute query
```

```
$qry_result = mysql_query($query) or die(mysql_error());

     //Build Result String
$display_string = "<table>";
$display_string .= "<tr>";
$display_string .= "<th>Name</th>";
$display_string .= "<th>Age</th>";
$display_string .= "<th>Sex</th>";
$display_string .= "<th>WPM</th>";
$display_string .= "</tr>";

// Insert a new row in the table for each person returned
while($row = mysql_fetch_array($qry_result)){
     $display_string .= "<tr>";
     $display_string .= "<td>$row[name]</td>";
     $display_string .= "<td>$row[age]</td>";
     $display_string .= "<td>$row[sex]</td>";
     $display_string .= "<td>$row[wpm]</td>";
     $display_string .= "</tr>";

}
echo "Query: " . $query . "<br />";
$display_string .= "</table>";
echo $display_string;
?>
```

Now enter a valid value in "Max Age" or any other box and then click Query MySQL button.



Your result will display here

If you have successfully completed this lesson, then you know how to use MySQL, PHP, HTML, and Javascript in tandem to write Ajax applications.

28. PHP – PHP and XML

XML is a markup language that looks a lot like HTML. An XML document is plain text and contains tags delimited by < and >. There are two big differences between XML and HTML:

- XML doesn't define a specific set of tags you must use.
- XML is extremely picky about document structure.

XML gives you a lot more freedom than HTML. HTML has a certain set of tags: the <a></a> tags surround a link, the <p> starts a paragraph and so on. An XML document, however, can use any tags you want. Put <rating></rating> tags around a movie rating, <height></height> tags around someone's height. Thus XML gives you option to device your own tags.

XML is very strict when it comes to document structure. HTML lets you play fast and loose with some opening and closing tags. But this is not the case with XML.

HTML list that's not valid XML

```
<ul>
<li>Braised Sea Cucumber
<li>Baked Giblets with Salt
<li>Abalone with Marrow and Duck Feet
</ul>
```

This is not a valid XML document because there are no closing </li> tags to match up with the three opening <li> tags. Every opened tag in an XML document must be closed.

HTML list that is valid XML

```
<ul>
<li>Braised Sea Cucumber</li>
<li>Baked Giblets with Salt</li>
<li>Abalone with Marrow and Duck Feet</li>
</ul>
```

Parsing an XML Document

PHP 5's new **SimpleXML** module makes parsing an XML document, well, simple. It turns an XML document into an object that provides structured access to the XML.

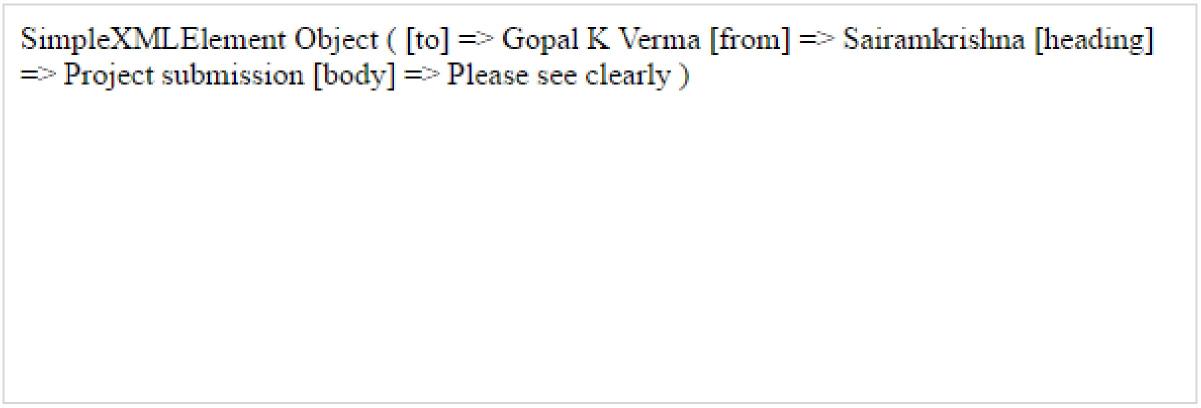
To create a SimpleXML object from an XML document stored in a string, pass the string to **simplexml_load_string( )**. It returns a SimpleXML object.

Example

Try out the following example:

```
<html>
   <body>

      <?php
         $note=<<<XML

         <note>
            <to>Gopal K Verma</to>
            <from>Sairamkrishna</from>
            <heading>Project submission</heading>
            <body>Please see clearly </body>
         </note>

         XML;
         $xml=simplexml_load_string($note);
         print_r($xml);
      ?>

   </body>
</html>
```

It will produce the following result:



```
SimpleXMLElement Object ( [to] => Gopal K Verma [from] => Sairamkrishna [heading]
=> Project submission [body] => Please see clearly )
```

NOTE: You can use function **simplexml_load_file(filename)** if you have XML content in a file.

For a complete detail of XML parsing function, check PHP Function Reference.

Generating an XML Document

SimpleXML is good for parsing existing XML documents, but you can't use it to create a new one from scratch.

The easiest way to generate an XML document is to build a PHP array whose structure mirrors that of the XML document and then to iterate through the array, printing each element with appropriate formatting.

Example

Try out the following example:

```
<?php


$channel = array('title' => "What's For Dinner",
                 'link' => 'http://menu.example.com/',
                 'description' => 'Choose what to eat tonight.');
print "<channel>\n";
foreach ($channel as $element => $content) {
   print " <$element>";
   print htmlentities($content);
   print "</$element>\n";
}
print "</channel>";
?>
```

It will produce the following result:

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

29. PHP – Object Oriented Programming

We can imagine our universe made of different objects like sun, earth, moon etc. Similarly, we can imagine our car made of different objects like wheel, steering, gear etc. In the same way, there are object oriented programming concepts which assume everything as an object and implement a software using different objects.

Object Oriented Concepts

Before we go in detail, let's define important terms related to Object Oriented Programming.

- **Class:** This is a programmer-defined datatype, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.
- **Object:** An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as **instance**.
- **Member Variable:** These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called **attribute** of the object once an object is created.
- **Member function:** These are the function defined inside a class and are used to access object data.
- **Inheritance:** When a class is defined by inheriting existing function of a parent class, then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.
- **Parent class:** A class that is inherited from by another class. This is also called a base class or super class.
- **Child Class:** A class that inherits from another class. This is also called a subclass or derived class.
- **Polymorphism:** This is an object oriented concept where the same function can be used for different purposes. For example, function name will remain same but it may take different number of arguments and can do different task.
- **Overloading:** a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly, functions can also be overloaded with different implementation.
- **Data Abstraction:** Any representation of data in which the implementation details are hidden (abstracted).
- **Encapsulation:** refers to a concept where we encapsulate all the data and member functions together to form an object.

- **Constructor:** refers to a special type of function which will be called automatically whenever there is an object formation from a class.

- **Destructors:** refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Defining PHP Classes

The general form for defining a new class in PHP is as follows:

```
<?php
class phpClass{
   var $var1;
   var $var2 = "constant string";
   function myfunc ($arg1, $arg2) {
      [..]
   }
   [..]
}
?>
```

Here is the description of each line:

- The special form **class**, followed by the name of the class that you want to define.

- A set of braces enclosing any number of variable declarations and function definitions.

- Variable declarations start with the special form **var**, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.

- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

Example

Here is an example which defines a class of Books type:

```
<?php
class  Books{
    /* Member variables */
    var $price;
    var $title;
    /* Member functions */
    function setPrice($par){
       $this->price = $par;
    }
    function getPrice(){
       echo $this->price ."<br/>";
    }
    function setTitle($par){
       $this->title = $par;
    }
    function getTitle(){
       echo $this->title ." <br/>";
    }
}
?>
```

The variable **$this** is a special variable and it refers to the same object, i.e., itself.

Creating Objects in PHP

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using **new** operator.

```
   $physics = new Books;
   $maths = new Books;
   $chemistry = new Books;
```

Here we have created three objects and these objects are independent of each other and they will have their existence separately. Next, we will see how to access member function and process member variables.

Calling Member Functions

After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only. Following example shows how to set title and prices for the three books by calling member functions.

```
   $physics->setTitle( "Physics for High School" );
   $chemistry->setTitle( "Advanced Chemistry" );
   $maths->setTitle( "Algebra" );

   $physics->setPrice( 10 );
   $chemistry->setPrice( 15 );
   $maths->setPrice( 7 );
```

Now you call another member functions to get the values set by in above example:

```
   $physics->getTitle();
   $chemistry->getTitle();
   $maths->getTitle();
   $physics->getPrice();
   $chemistry->getPrice();
   $maths->getPrice();
```

This will produce the following result:

```
Physics for High School
Advanced Chemistry
Algebra
10
15
7
```

Constructor Functions

Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behavior, by initializing many things through constructor functions.

PHP provides a special function called **__construct()** to define a constructor. You can pass as many as arguments you like into the constructor function.

Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

```
function __construct( $par1, $par2 ){
   $this->price = $par1;
   $this->title = $par2;
}
```

Now we don't need to call set function separately to set price and title. We can initialize these two member variables at the time of object creation only. Check following example below:

```
   $physics = new Books( "Physics for High School", 10 );
   $maths = new Books ( "Advanced Chemistry", 15 );
   $chemistry = new Books ("Algebra", 7 );

   /* Get those set values */
   $physics->getTitle();
   $chemistry->getTitle();
```

```
    $maths->getTitle();


    $physics->getPrice();
    $chemistry->getPrice();
    $maths->getPrice();
```

This will produce the following result:

```
  Physics for High School
  Advanced Chemistry
  Algebra
  10
  15
  7
```

Destructor

Like a constructor function you can define a destructor function using function **__destruct()**. You can release all the resources with-in a destructor.

Inheritance

PHP class definitions can optionally inherit from a parent class definition by using the extends clause. The syntax is as follows:

```
  class Child extends Parent {
     <definition body>
  }
```

The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics:

- Automatically has all the member variable declarations of the parent class.
- Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.

Following example inherit Books class and adds more functionality based on the requirement.

```
class Novel extends Books{
   var publisher;
   function setPublisher($par){
     $this->publisher = $par;
   }
   function getPublisher(){
     echo $this->publisher. "<br />";
   }
}
```

Now apart from inherited functions, class Novel keeps two additional member functions.

Function Overriding

Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.
In the following example, getPrice and getTitle functions are overriden to retrun some values.

```
    function getPrice(){
       echo $this->price . "<br/>";
       return $this->price;
    }
    function getTitle(){
       echo $this->title . "<br/>";
       return $this->title;
    }
```

Public Members

Unless you specify otherwise, properties and methods of a class are public. That is to say, they may be accessed in three possible situations:

- From outside the class in which it is declared
- From within the class in which it is declared
- From within another class that implements the class in which it is declared

Till now we have seen all members as public members. If you wish to limit the accessibility of the members of a class, then you define class members as **private** or **protected**.

Private members

By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.

A class member can be made private by using **private** keyword in front of the member.

```
class MyClass {
   private $car = "skoda";
   $driver = "SRK";

   function __construct($par) {
      // Statements here run every time an instance of the class is created.
   }

   function myPublicFunction() {
      return("I'm visible!");
   }
   private function myPrivateFunction() {
      return("I'm  not visible outside!");
   }
}
```

When *MyClass* class is inherited by another class using extends, myPublicFunction() will be visible, as will $driver. The extending class will not have any awareness of or access to myPrivateFunction and $car, because they are declared private.

Protected members

A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class. Protected members are not available outside of those two kinds of classes. A class member can be made protected by using **protected** keyword in front of the member.

Here is different version of MyClass:

```
class MyClass {
   protected $car = "skoda";
   $driver = "SRK";

   function __construct($par) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }
   function myPublicFunction() {
      return("I'm visible!");
   }
   protected function myPrivateFunction() {
      return("I'm  visible in child class!");
   }
}
```

Interfaces

Interfaces are defined to provide a common function names to the implementors. Different implementors can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.

As of PHP5, it is possible to define an interface, like this:

```
interface Mail {
   public function sendMail();
}
```

Then, if another class implemented that interface, like this:

```
class Report implements Mail {
   // sendMail() Definition goes here
}
```

Constants

A constant is somewhat like a variable, in that it holds a value, but is really more like a function because a constant is immutable. Once you declare a constant, it does not change. Declaring one constant is easy, as is done in this version of MyClass:

```
class MyClass {
   const requiredMargin = 1.7;
   function __construct($incomingValue) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }
}
```

In this class, requiredMargin is a constant. It is declared with the keyword const, and under no circumstances can it be changed to anything other than 1.7. Note that the constant's name does not have a leading $, as variable names do.

Abstract Classes

An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword **abstract**, like this:

When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.

```
abstract class MyAbstractClass {
   abstract function myAbstractFunction() {
   }
}
```

Note that the function definitions inside an abstract class must also be preceded by the keyword abstract. It is not legal to have abstract function definitions inside a non-abstract class.

Static Keyword

Declaring class members or methods as static makes them accessible without needing an instantiation of the class. A member declared as static cannot be accessed with an instantiated class object (though a static method can).

Try out the following example:

```
<?php
class Foo
{
    public static $my_static = 'foo';

    public function staticValue() {
        return self::$my_static;
    }
}
print Foo::$my_static . "\n";
$foo = new Foo();
print $foo->staticValue() . "\n";
```

Final Keyword

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final, then it cannot be extended.

The following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

```
<?php
class BaseClass {
   public function test() {
       echo "BaseClass::test() called<br>";
   }




   final public function moreTesting() {
       echo "BaseClass::moreTesting() called<br>";
   }
}

class ChildClass extends BaseClass {
   public function moreTesting() {
       echo "ChildClass::moreTesting() called<br>";
   }
}
?>
```

Calling parent constructors

Instead of writing an entirely new constructor for the subclass, let's write it by calling the parent's constructor explicitly and then doing whatever is necessary in addition for instantiation of the subclass. Here's a simple example:

```
class Name
{
   var $_firstName;
   var $_lastName;
   function Name($first_name, $last_name)
   {
     $this->_firstName = $first_name;
     $this->_lastName = $last_name;
   }
   function toString() {
     return($this->_lastName .", " .$this->_firstName);
   }
}
class NameSub1 extends Name
{
   var $_middleInitial;
```

```
    function NameSub1($first_name, $middle_initial, $last_name) {
        Name::Name($first_name, $last_name);
        $this->_middleInitial = $middle_initial;
    }



    function toString() {
        return(Name::toString() . " " . $this->_middleInitial);
    }
}
```

In this example, we have a parent class (Name), which has a two-argument constructor, and a subclass (NameSub1), which has a three-argument constructor. The constructor of NameSub1 functions by calling its parent constructor explicitly using the :: syntax (passing two of its arguments along) and then setting an additional field. Similarly, NameSub1 defines its nonconstructor toString() function in terms of the parent function that it overrides.

NOTE: A constructor can be defined with the same name as the name of a class. It is defined in above example.

30. PHP – PHP for C Developers

The simplest way to think of PHP is as interpreted C that you can embed in HTML documents. The language itself is a lot like C, except with untyped variables, a whole lot of Web-specific libraries built in, and everything hooked up directly to your favorite Web server.
The syntax of statements and function definitions should be familiar, except that variables are always preceded by $, and functions do not require separate prototypes.
Here we will put some similarities and differences in PHP and C

Similarities

- **Syntax:** Broadly speaking, PHP syntax is the same as in C: Code is blank insensitive, statements are terminated with semicolons, function calls have the same structure (my_function(expression1, expression2)), and curly braces ({ and }) make statements into blocks. PHP supports C and C++-style comments (/* */ as well as //), and also Perl and shell-script style (#).

- **Operators:** The assignment operators (=, +=, *=, and so on), the Boolean operators (&&, ||, !), the comparison operators (<,>, <=, >=, ==, !=), and the basic arithmetic operators (+, -, *, /, %) all behave in PHP as they do in C.

- **Control structures:** The basic control structures (if, switch, while, for) behave as they do in C, including supporting break and continue. One notable difference is that switch in PHP can accept strings as case identifiers.

- **Function names:** As you peruse the documentation, you.ll see many function names that seem identical to C functions.

Differences

- **Dollar signs:** All variables are denoted with a leading $. Variables do not need to be declared in advance of assignment, and they have no intrinsic type.

- **Types:** PHP has only two numerical types: integer (corresponding to a long in C) and double (corresponding to a double in C). Strings are of arbitrary length. There is no separate character type.

- **Type conversion:** Types are not checked at compile time, and type errors do not typically occur at runtime either. Instead, variables and values are automatically converted across types as needed.

- **Arrays:** Arrays have a syntax superficially similar to C's array syntax, but they are implemented completely differently. They are actually associative arrays or hashes, and the index can be either a number or a string. They do not need to be declared or allocated in advance.

- **No structure type:** There is no struct in PHP, partly because the array and object types together make it unnecessary. The elements of a PHP array need not be of a consistent type.

- **No pointers:** There are no pointers available in PHP, although the typeless variables play a similar role. PHP does support variable references. You can also emulate function pointers to some extent, in that function names can be stored in variables and called by using the variable rather than a literal name.

- **No prototypes:** Functions do not need to be declared before their implementation is defined, as long as the function definition can be found somewhere in the current code file or included files.

- **Memory management:** The PHP engine is effectively a garbage-collected environment (reference-counted), and in small scripts there is no need to do any deallocation. You should freely allocate new structures - such as new strings and object instances. IN PHP5, it is possible to define destructors for objects, but there is no free or delete. Destructors are called when the last reference to an object goes away, before the memory is reclaimed.

- **Compilation and linking:** There is no separate compilation step for PHP scripts.

- **Permissiveness:** As a general matter, PHP is more forgiving than C (especially in its type system) and so will let you get away with new kinds of mistakes. Unexpected results are more common than errors.

31. PHP – PHP for PERL Developers

This chapter will list out major similarities and differences in between PHP and PERL. This will help PERL developers to understand PHP very quickly and avoid common mistakes.

Similarities

- **Compiled scripting languages:** Both Perl and PHP are scripting languages. This means that they are not used to produce native standalone executables in advance of execution.

- **Syntax:** PHP's basic syntax is very close to Perl's, and both share a lot of syntactic features with C. Code is insensitive to whitespace, statements are terminated by semicolons, and curly braces organize multiple statements into a single block. Function calls start with the name of the function, followed by the actual arguments enclosed in parentheses and separated by commas.

- **Dollar-sign variables:** All variables in PHP look like scalar variables in Perl: a name with a dollar sign ($) in front of it.

- **No declaration of variables:** As in Perl, you don.t need to declare the type of a PHP variable before using it.

- **Loose typing of variables:** As in Perl, variables in PHP have no intrinsic type other than the value they currently hold. You can store either number or string in same type of variable.

- **Strings and variable interpolation:** Both PHP and Perl do more interpretation of double-quoted strings ("string") than of single-quoted strings ('string').

Differences

- **PHP is HTML-embedded:** Although it is possible to use PHP for arbitrary tasks by running it from the command line, it is more typically connected to a Web server and used for producing Web pages. If you are used to writing CGI scripts in Perl, the main difference in PHP is that you no longer need to explicitly print large blocks of static HTML using print or heredoc statements and instead can simply write the HTML itself outside of the PHP code block.

- **No @ or % variables:** PHP has one only kind of variable, which starts with a dollar sign ($). Any of the datatypes in the language can be stored in such variables, whether scalar or compound.
- **Arrays versus hashes:** PHP has a single datatype called an array that plays the role of both hashes and arrays/lists in Perl.

- **Specifying arguments to functions:** Function calls in PHP look pretty much like subroutine calls in Perl. Function definitions in PHP, on the other hand, typically require some kind of list of formal arguments as in C or Java which is not the csse in PERL.

- **Variable scoping in functions:** In Perl, the default scope for variables is global. This means that top-level variables are visible inside subroutines. Often, this leads to promiscuous use of globals across functions. In PHP, the scope of variables within function definitions is local by default.

- **No module system as such:** In PHP there is no real distinction between normal code files and code files used as imported libraries.

- **Break and continue rather than next and last:** PHP is more like C language and uses break and continue instead of next and last statement.

- **No elsif:** A minor spelling difference: Perl's elsif is PHP's elseif.

- **More kinds of comments:** In addition to Perl-style (#) single-line comments, PHP offers C-style multiline comments (/* comment */) and Java-style single-line comments (// comment).

- **Regular expressions:** PHP does not have a built-in syntax specific to regular expressions, but has most of the same functionality in its "Perl-compatible" regular expression functions.

Part 3: Function Reference

32. PHP – Function Reference

PHP is very rich in terms of Built-in functions. Here is the list of various important function categories. There are various other function categories which are not covered here. Select a category to see a list of all the functions related to that category.

- PHP Array Functions
- PHP Calendar Functions
- PHP Class/Object Functions
- PHP Character Functions
- PHP Date & Time Functions
- PHP Directory Functions
- PHP Error Handling Functions
- PHP File System Functions
- PHP MySQL Functions
- PHP Network Functions
- PHP ODBC Functions
- PHP String Functions
- PHP SimpleXML Functions
- PHP XML Parsing Functions

Link to other Categories of PHP Functions

- PHP Functions Manual