PHP Programming

INDEX

What is PHP?

PHP is a scripting language designed to fill the gap between SSI (Server Side Includes) and Perl, intended largely for the web environment. PHP has gained quite a following in recent times, and it is one of the forerunners in the Open Source software movement. Its popularity derives from its C-like syntax, its speed and its simplicity. PHP is currently divided into two major versions: PHP 4 and PHP 5, although PHP 4 is deprecated and is no longer developed or supplied with critical bug fixes. PHP 6 is currently under development.

If you've ever been to a website that needs you to login, you've probably encountered a server-side scripting language. Due to its market saturation, this means you've probably come across PHP. PHP was designed by Rasmus Lerdorf to display his resume online and to collect data from his visitors.

Basically, PHP allows a static web document to become dynamic. "PHP" is a recursive acronym that stands for "PHP: Hypertext Preprocessor". PHP preprocesses (that is, PHP processes before the output is sent to the browser) hypertext documents. Because of this, the pages can change before the user sees them, based on conditions. This can be used to write something to the page, create a table with a number of rows equal to the number of times the user has visited, or integrate the web page with a web database, such as MySQL.

Before you embark on the wonderful journey of Server Side Processing, it is recommended that you have some basic understanding of the HyperText Markup Language. PHP is also being used to build GUI-driven applications; PHP-GTK is used to build Graphical User Interfaces.

PHP Programming/Setup and Installation

Setup and Installation

Since PHP is a server-side technology, you should naturally expect to invest some time in setting up a server environment for production, development or learning. To be frank, PHP is quite easy to set up compared to other monsters like J2EE. Nevertheless, the procedures are complicated by the various combinations of different versions of web server, PHP and database (most often MySQL). Below I will introduce the steps needed to set up a working PHP environment with MySQL database.

Linux

If your desktop runs on Linux, chances are that Apache, PHP, and MySQL are already installed for you. This wildly popular configuration is commonly referred to as LAMP, i.e. Linux Apache MySQL PHP, or P, the latter 'P', can also refer to Perl another major player in the opensource web service arena. If some components are not installed, you will likely have to manually install the following packages:

Apache or Lighttpd
PHP
MySQL or Postgres
The PHP integration plugin for the database.

Debian or its derivatives

On Debian or its derivatives, Ubuntu included[1], you can use the corresponding commands:
apt-get install php5

Server
If you wish to use Apache
apt-get install apache2
-or-

If you wish to use Lighttpd
apt-get install lighttpd

Database
If you wish to use Postgres
apt-get install postgres-server postgres-client php5-pg
-or-
If you wish to use Mysql
apt-get install mysql-server mysql-client php5-mysql

^ If you chose to use Ubuntu with Apache and MySQL you might wish to utilize the Ubuntu community site for such a configuration ubuntu lamp wiki.

Gentoo

For Gentoo Linux users, the gentoo-wiki has this HowTo available: Apache2 with PHP and MySQL.

In general, you'll want to do the following under Gentoo:
emerge apache
emerge mysql
emerge mod_php

RPM-based

The exact procedures depend on your Linux distribution. On a Fedora system, the commands are typically as follows:
yum install httpd
yum install php
yum install mysql
yum install php-mysql

It's impossible to cover all the variants here, so consult your Linux distribution's manual for more details, or grab a friend to do it for you.

One sure-fire way of getting PHP up and running on your *nix system is to compile it from source. This isn't as hard as it may sound and there are good instructions available in the PHP manual.

Windows

Contrary to what some people may think, PHP on Windows is a very popular option. On a Windows platform, you have the option to use either the open source Apache web server, or the native Internet Information Services (IIS) server from Microsoft, which can be installed from your Windows CD. When you have one of these servers installed, you can download and install the appropriate PHP Windows binaries distributions from PHP download page. The installer version requires less user-interaction.

For increased performance you will want to use FastCGI. There is a wikibook that will assist you on Setting up IIS with FastCGI.

Databases

On Microsoft Windows you must always install your own database. Two popular choices are the open source Postgres, and MySQL. Postgres is more liberally licensed, and is free to use for commercial purposes.

Postgresql

Official Zend documentation: http://us.php.net/pgsql

Postgres is simple and easy to install, browse to http://www.postgresql.org/ftp/binary/v8.3.0/win32/ and download the exe and double-click.

MySQL

Official MySQL documentation: http://us.php.net/mysql

You might wish to install the MySQL database. You can download the Windows version of MySQL, and follow the installation instructions. If you have PHP 4, you do not need to install the equivalence of php-mysql on Linux, as MySQL support is built-in in Windows distributions of PHP. In PHP 5 you will need to uncomment the following line in your php.ini file (that is, remove the ';' at the beginning of the line):

```
;extension=php_mysql.dll
```

Bundled Package

If you find all the above too much a hassle, you have another option. Driven by the eternal desire to do things the safe/easy way, several conveniently packaged AMP bundles of Apache/MySQL/PHP can be found on the net. One of them is PHPTriad. Or, you can try Uniform Server. It is a small WAMP Package. 1 click install and also easy to use. Same with XAMPP for Windows. And after trying these out you can simply delete the directory and everything is clean. A number of portable Windows AMP package choices are summarized at List of portable Web Servers.

Also, a package installer called WAMP is available. It simply installs Apache, PHP and MySQL on windows with ease. http://www.en.wampserver.com/

Mac OS X

Mac OS X comes with Apache server as standard, and enabling it is as simple as checking the box next to 'Personal Web Sharing' in the 'Sharing' section of System Preferences. Once you have done this you can place files in /Library/WebServer/Documents to access them on your server. Mac OS X does come with PHP but the installation lacks any significant quantity of extensions, so if you want any you're going to have to install PHP yourself. You can do this by following the instructions in Apple's Developer Connection, or you can download an automatic installer such as the ones available at Entropy. Once you've done one of those, you'll have a server with PHP running on your Mac.

To install MySQL just download and run the OS X installer package or use XAMPP for MacOS X.

If you use unix or learning it, however, compiling might be the way to go for all three, or just the ones you like. The advantage is that you can choose exactly which extensions you

want for PHP and Apache. Also you can choose which versions to compile together. To do this make sure you have the Developer Tools installed. They are shipped with OS X.

How Do I Know My Setup is Working?

After you have successfully completed the previous section, it's time to make sure that everything went well. You also get the chance to write your very first PHP scripts! Open your favourite plain text editor (not Microsoft Word or another word processor), and type the following magical line:

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

Save it as phpinfo.php in your web server's root document directory. If you are using a web hosting server, upload it to the server to where you would place HTML files. Now, open up your web browser, and go to http://localhost/phpinfo.php, or http://your-web-hosting-server.com/phpinfo.php if you are using a web hosting server, and look at the output.

Now scroll down that page and make sure there is a table with the title "mysql", and the top row should read: "MySQL support: enabled". If your output does not have this, your particular installation of PHP does not have MySQL support enabled. Note that this test doesn't tell you whether MySQL server is running. You should fire up your MySQL client and check before you proceed.

Some dedicated php or script editors even have color coding of different words which can be very useful for finding mistakes. A free implementation of this is the powerful Notepad++, available from Sourceforge and licensed under the GPL.

Hello World

"Hello world." is the first program most beginning programmers will learn to write in any given language. Here is an example of how to print "Hello world!" in PHP.

Code:

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

Output:

```
Hello world!
```

This is as basic as PHP gets. Three simple lines, the first line identifies that everything beyond the <?php tag, until the ?> tag, is PHP code. The second line causes the greeting to be printed (or echoed) to the web page. This next example is slightly more complex and uses variables.

Hello World With Variables

This example stores the string "Hello world!" in a variable called $string. The following lines show various ways to display the variable $string to the screen.

PHP Code:

```
<?php

  // Declare the variable 'string' and assign it a value.
  // The <br /> is the HTML equivalent to a new line.
  $string = 'Hello world!<br />';

  // You can echo the variable, similar to the way you would echo a string.
  echo $string;

  // You could also use print.
  print $string;

  // Or, if you are familiar with C, printf can be used too.
  printf('%s', $string);
?>
```

PHP Output:

```
Hello world!<br />Hello world!<br />Hello world!<br />
```

HTML Render:

Hello world!
Hello world!
Hello world!

The previous example contained two outputs. PHP can output HTML that your browser will format and display. The PHP Output box is the exact PHP output. The HTML Render box is approximately how your browser would display that output. Don't let this confuse you, this is just to let you know that PHP can output HTML. We will cover this much more in depth later.

New Concepts

Variables

Variables are the basis of any programming language: they are "containers" (spaces in memory) which hold data. The data can change, thus it is "variable".

If you've had any experience with other programming languages, you know that in some of the languages, you must define the type of data that the variable will hold. Those languages are called statically-typed, because the types of variables must be known before you store something in them. Programming languages such as C++ and Java are statically-typed. PHP, on the other hand, is dynamically-typed, because the type of the variable is linked to the value of the variable. You could define a variable for a string, store a string, and then replace the string with a number. To do the same thing in C++, you would have to cast, or change the type of, the variable, and store it in a different "container".

All variables in PHP follow the format of a dollar sign ($) followed by an identifier i.e. $variable_name. These identifiers are case-sensitive, meaning that capitalization matters, so $wiki is different from $Wiki.

Real world analogy

To compare a variable to real world objects, imagine your computer's memory as a storage shed. A variable would be a box in that storage shed and the contents of the box (such as a cup) would be the data in that variable.

If the box was labeled kitchen stuff and the box's contents were a cup, the PHP code would be:

```
$kitchen_stuff = 'cup';
```

If I then went into the storage shed, opened the box labeled kitchen stuff, and then replaced the cup with a fork, the new code would be:

```
$kitchen_stuff = 'fork';
```

Notice the addition of the = in the middle and the ; at the end of the code block. The = is the assignment operator, or in our analogy, instructions that came with the box that states "put the cup in the box". The ; indicates to stop evaluating the block of code, or in our analogy, finish up with what you are doing and move on to something else.

Also notice the cup was wrapped in single quotes instead of double. Using double quotes would tell the PHP parser that there may be more than just a cup going into the box and to look for additional instructions.

```
$bathroom_stuff = 'toothbrush';
$kitchen_stuff = "cup $bathroom_stuff";

//$kitchen_stuff contents is now "'cup toothbrush"'
```

Single quotes tell the PHP parser that it's only a cup and to not look for anything more. In this example the bathroom box that should've had its contents added to the kitchen box has its name added instead.

```
$bathroom_stuff = 'toothbrush';
$kitchen_stuff = 'cup $bathroom_stuff';

//$kitchen_stuff contents is now "'cup $bathroom_stuff"'
```

So again, try to visualize and associate the analogy to grasp the concept of variables with the comparison below. Note that this is a real world object comparison and NOT PHP code.

Computer memory (RAM) = storage shed
Variable = a box to hold stuff
Variable name = a label on the box such as kitchen stuff
Variable data = the contents of the box such as a cup

Notice that you wouldn’t name the variable box, as the relationship between the variable and the box is represented by the code>$ and how the data is stored in memory. For example a constant and array can be considered a type of variable when using the box analogy as they all are containers to hold some sort of contents, however, the difference is on how they are defined to handle the contents in the box.

Variable: a box that can be opened while in the storage shed to exchange the contents in the box.

Constant: a box that cannot be opened to exchange its contents. Its contents can only be viewed and not exchanged while inside the storage shed.

Array: a box that contains 1 or more additional boxes in the main box. To complicate matters for beginners, each additional box may contain a box as well. In the kitchen stuff box we have two boxes, the clean cup box

```
$kitchen_stuff["clean_cup"] = 'the clean cup';
```

and the dirty cup box

```
$kitchen_stuff["dirty_cup"] = 'the dirty cup';
```

More on variables, from the PHP manual

The print and echo statements

Print is the key to output. It sends whatever is in the quotes (or parentheses) which follow it to the output device (browser window). A similar function is echo, but print allows the user to check whether or not the print succeeded.
When used with quotation marks, as in:

```
print "Hello, World!";
```

The quoted text is treated as if it were a string, and thus can be used in conjunction with the concatenation (joining two strings together) operator as well as any function that returns a string value.

The following two examples have the same output.
print "Hello, World!";

and
print "Hello" . ", " . "World!";

The dot symbol concatenates two strings. In other programming languages, concatenating a string is done with the plus symbol and the dot symbol is generally used to call functions from classes.

Also, it might be useful to note that under most conditions echo can be used interchangably with print. print returns a value, so it can be used to test if the print succeeded, while echo assumes everything worked. Under most conditions there is nothing we can do if echo fails.

The following examples have the same output again.
echo "Hello, World!";

and
echo "Hello" . ", " . "World!";

We will use echo in most sections of this book, since it is the more commonly used statement.

It should be noted that while echo and print can be called in the same way as functions, they are, in fact, language constructs, and can be called without the brackets. Normal functions (almost all others) must be called with brackets following the function identifier.

BASICS

The Examples

Example 1 - Basic arithmetic operators

This example makes use of the five basic operators used in mathematical expressions. These are the foundation of all mathematical and string operations performed in PHP.

The five mathematical operators all function identically to those found in C++ and Java
add (+)
subtract (-)
multiply (*)
divide (/)
assign (=)

Examine this example. Each mathematical expression to the right of the assign operator is evaluated, using the normal order of operations. When the expression has been evaluated, the resultant value is assigned to the variable named to the left of the assign operator.

PHP Code:

```
<?php
  $x = 25;
  $y = 10;
  $z = $x + $y;
  echo $z;

  echo "<br />";
  $z = $x / $y;
  echo $z;

  echo "<br />";
  $z = $y * $y * $x;
  echo $z - 1250;
  echo "<br />";
?>
```

PHP Output:

```
35<br />2.5<br />1250<br />
```

HTML Render:

35
2.5
1250

Note: If you are not familiar with (X)HTML, you may not know the purpose of this part of the above code:
echo "
";
Its purpose is to insert an HTML "line break" between the results, causing the browser to

display each result on a new line when rendering the page. In the absence of this line, the above code would instead print: 352.51250

This is of course not the desired result.

There are two code options which perform the opposite of the assign (=) operator. The keyword null should be used for variable nullification, which is actually used with the assign (=) operator in place of a value. If you want to destroy a variable, the unset() language construct is available.

Examples:
$variable = null;

or
unset($variable);

Example 2 - String concatenation

This example demonstrates the concatenation operator (.), which joins together two strings, producing one string consisting of both parts. It is analogous to the plus (+) operator commonly found in C++ string class (see STL), Python, Java, JavaScript implementations.

The statement
$string = $string . " " . "All the cool kids are doing it.";
prepends the current value of $string (which is "PHP is wonderful and great.") to the literal string " All the cool kids are doing it." and assigns this new string to $string.

Code:

```
<?php
  $string = "PHP is wonderful and great.";
  $string = $string . " " . "All the cool kids are doing it.";
  echo $string;
?>
```

Output:
PHP is wonderful and great. All the cool kids are doing it.

Example 3 - Shortcut operators

This snippet demonstrates self-referential shortcut operators. The first such operator is the ++ operator, which increments $x (using the postfix form) by 1 giving it the value 2. After incrementing $x, $y is defined and assigned the value 5.

The second shortcut operator is *=, which takes $y and assigns it the value $y * $x, or 10.

After initializing $z to 180, the subsequent line performs two shortcut operations. Going by order of operations (see manual page below), $y is decremented (using the prefix form) and divided into $z. $z is assigned to the resulting value, 20.

Code:

```
<?php
  $x = 1;
  $x++;
  echo $x . " ";
  $y = 5;
  $y *= $x;
  echo $y . " ";
  $z = 180;
  $z /= --$y;
  echo $z;
?>
```

Output:
2 10 20

Note: The expanded version of the above code (without the shortcut operators) looks like this:

```
<?php
  $x = 1;
  $x = $x + 1;
  echo $x . " ";
  $y = 5;
  $y = $y * $x;
  echo $y . " ";
  $z = 180;
  $y = $y - 1;
  $z = $z / $y;
  echo $z;
?>
```

The output is the same as seen in the above example.

New Concepts

Operators

An operator is any symbol used in an expression used to manipulate data. The seven basic PHP operators are:
= (assignment)
+ (addition)
- (subtraction)
* (multiplication)
/ (division)
% (modulus)
. (concatenation)

In addition, each of the above operators can be combined with an assignment operation, creating the operators below:
+= (addition assignment)
-= (subtraction assignment)
*= (multiplication assignment)
/= (division assignment)

%= (modulus assignment)
.= (concatenation assignment)

These operators are used when a variable is added, subtracted, multiplied or divided by a second value and subsequently assigned to itself.
In other words, the statements
$var = $var + 5;

and
$var += 5;
are equivalent.

There are also increment and decrement operators in PHP.
++ (increment)
-- (decrement)

These are a special case of the addition and subtraction assignment operators.
This code uses the addition assignment operator to increment and decrement a variable.

Code:
```
$var = 0;

$var += 1;
echo "The incremented value is $var.\n";

$var -= 1;
echo "The decremented value is $var.\n";
```

Output:
```
The incremented value is 1.
The decremented value is 0.
```

While this is perfectly legal in PHP, it is somewhat lengthy for an operation as common as this. It can easily be replaced by the increment operator, shortening the statement.
This code snippet uses the increment and decrement operators to increase and decrease a variable's value by one.

Code:
```
$var = 3;

$var++;
echo "The incremented value is $var.\n";

$var--;
echo "The decremented value is $var.\n";
```

Output:
```
The incremented value is 4.
The decremented value is 3.
```
Using the increment operator makes your code slightly easier to read and understand.

For a more in-depth overview of PHP's operators, including an explanation of bitwise operators, refer to the manual link below.

Newline and Other Special Characters

Both of the below examples make use of the newline character (\n) to signify the end of the current line and the beginning of a new one.

The newline is used as follows:

Code:
```
echo "PHP is cool,\nawesome,\nand great.";
```

Output:
```
PHP is cool,
awesome,
and great.
```

Notice: the line break occurs in the output wherever the \n occurs in the string in the echo statement. However, a \n does not produce a newline when the HTML document is displayed in a web browser. This is because the PHP engine does not render the script. Instead, the PHP engine outputs HTML code, which is subsequently rendered by the web browser. The linebreak \n in the PHP script becomes HTML whitespace, which is skipped when the web browser renders it (much like the whitespace in a PHP script is skipped when the PHP engine generates HTML). This does not mean that the \n operator is useless; it can be used to add whitespace to your HTML, so if someone views the HTML generated by your PHP script they'll have an easier time reading it.

In order to insert a line-break that will be rendered by a web browser, you must instead use the
 tag to break a line.

Therefore the statement above would be altered like so:
```
echo 'PHP is cool,<br />awesome<br />and great.';
```

The function nl2br() is available to automatically convert newlines in a string to
 tags. The string must be passed through the function, and then reassigned:

PHP Code:
```
$string = "This\ntext\nbreaks\nlines.";
$string = nl2br($string);
print $string;
```

PHP Output:
```
This<br />
text<br />
breaks<br />
lines.
```

HTML Render:
This
text
breaks
lines.
Additionally, the PHP output (HTML source code) generated by the above example includes linebreaks.

Other special characters include the ASCII NUL (\0) - used for padding binary files, tab (\t) - used to display a standard tab, and return (\r) - signifying a carriage return. Again, these characters do not change the rendering of your HTML since they add whitespace to the HTML source. In order to have tabs and carriage returns rendered in the final web page, &tab; should be used for tabs and
 should be used for a carriage return.

Input to PHP

PHP has a set of functions that retrieve input. If you are using stanard input (such as that from a command-line), it is retrieves using the basic input functions:
Reading from standard input:
$mystring = fgets($stdin);

Or:
$stdin = fopen('php://stdin', 'r'); // opens standard input
$line = fgets($stdin); // reads until user presses ENTER

Webservers

On webservers, information sent to a PHP app may either be a GET operation or a POST operation.

For a GET operation, the parameters are sent through the address bar. Parameters within the bar may be retrieves by using accessing $_GET['parameter']. On a POST operation, submitted input is accessed by $_POST['parameter'].

A more generic array, $_REQUEST['parameter'] contains the contents of both $_GET, $_POST, and $_COOKIE.

Comments

Introduction

As you write more complex scripts, you'll see that you must make it clear to yourself and to others exactly what you're doing and why you're doing it. Comments and "good" naming can help you make clear and understandable scripts because:
When writing a script takes longer than a week, by the time you're done, you won't remember what you did when you started, and you will most likely need to know.
Any script that is commonly used will need rewriting sooner or later. Rewriting is much easier (and in many cases, made possible) when you write down what you did.
If you want to show someone your scripts, they should be nice and neat.

Comments

Comments are pieces of code that the PHP parser skips. When the parser spots a comment, it simply keeps going until the end of the comment without doing anything. PHP offers both one line and multi-line comments.

One-Line Comments

One-line comments are comments that start where ever you start them and end at the end of the line. With PHP, you can use both // and # for your one-line comments (# is not commonly used). Those are used mainly to tell the reader what you're doing the next few lines. For example:

```
 //Print the variable $message
 echo $message;
```

It's important to understand that a one-line comment doesn't have to 'black out' the whole line, it starts where ever you start it. So it can also be used to tell the reader what a certain variable does:

```
 $message = ""; //This sets the variable $message to an empty string
```

The $message = ""; is executed, but the rest of the line is not.

Multi-Line Comments

This kind of comment can go over as many lines as you'd like, and can be used to state what a function or a class does, or just to contain comments too big for one line. To mark the beginning of a multiline comment, use /* and to end it, use */ . For example:

```
/* This is a
    multiline comment
   And it will close
    When I tell it to.
  */
```

You can also use this style of comment to skip over part of a line. For example:

```
  $message = ""/*this would not be executed*/;
```

Although it is recommended that one does not use this coding style, as it can be confusing in some editors.

Naming

Naming your variables, functions and classes correctly is very important. If you define the following variables:

```
$var1 = "PHP";
$var2 = 15;
```

They won't say much to anyone. But if you do it like this:

```
$programming_language = "PHP";
$menu_items = 15;
```

It would be much clearer. But don't go too far. $programming_language, for example is not a good name. It's too long, and will take you a lot of time to type and can affect clarity. A better name may be $prog_language, because it's shorter but still understandable. Don't forget to use comments as well, to mark what every variable does.

```
$prog_language = "PHP";  //The programming language used to write this script
$menu_items = 15;        //The maximum number of items allowed in your personal menu
```

Magic numbers

When using numbers in a program it is important that they have a clear meaning. For instance it's better to define $menu_items early on instead of using 15 repeatedly without telling people what it stands for. The major exception to this is the number 1; often programmers have to add or subtract 1 from some other number to avoid off-by-one errors, so 1 can be used without definition.

When you define numbers early on in their usage it also makes it easier to adjust the values later. Again if we have 15 menu items and we refer to them ten times, it will be a lot easier to adjust the program when we add a 16th menu item; just correct the variable definition and you have updated the code in 10 places.

Spacing

PHP ignores extra spaces and lines. This means, that even though you could write code like this:

```
if($var == 1) {echo "Good";} else {echo "Bad";}
```

It's better like this:

```
if($var == 1) {
  echo "Good";
} else {
  echo "Bad";
}
```

Some programmers prefer this way of writing:

```
if($var == 1)
{
  echo "Good";
```

```
}
else
{
  echo "Bad";
}
```

You should also use blank lines between different portions of your script. Instead of

```
$var = 1;
echo "Welcome!&lt;br /&gt;";
echo "How are you today?&lt;br /&gt;";
echo "The answer: ";
if($var == 1) {
  echo "Good";
} else {
  echo "Bad";
}
```

You could write:

```
$var = 1;

echo "Welcome!&lt;br /&gt;";
echo "How are you today?&lt;br /&gt;";

echo "The answer: ";
if($var == 1) {
  echo "Good";
} else {
  echo "Bad";
}
```

And the reader will understand that your script first declares a variable, then welcomes the user, and then checks the variable.

Arrays

Arrays

Arrays are sets of data which can be defined in a PHP Script. Arrays can contain other arrays inside of them without any restriction (hence building multidimensional arrays). Arrays can be referred to as tables or hashes.

Syntax

Arrays can be created in two ways. The first involves using the function array. The second involves using square brackets.

The array function method

In the array function method, you create a array in the scheme of:
$foo = bar()

For example, to set the array up to make the keys sequential numbers (Example: "0, 1, 2, 3"), you use:
```
$foobar = array($foo, $bar);
```

This would produce the array like this:
```
$foobar[0] = $foo
$foobar[1] = $bar
```

It is also possible to define the key value:
```
$foobar = array('foo' => $foo, 'bar' => $bar);
```

This would set the array like this:
```
$foobar['foo'] = $foo
$foobar['bar'] = $bar
```

The square brackets method

The square brackets method allows you to set up by directly setting the values. For example, to make $foobar[1] = $foo, all you need to do is:
```
$foobar[1] = $foo;
```

The same applies for setting the key value:
```
$foobar['foo'] = $foo;
```

Examples of Arrays

Example #1
This example sets and prints arrays.

PHP Code:
```
<?php
```

```
 $array =
array("name"=>"Toyota","type"=>"Celica","colour"=>"black","manufactured"=>"1991");
 $array2 = array("Toyota","Celica","black","1991");
 $array3 = array("name"=>"Toyota","Celica","colour"=>"black","1991");
 print_r($array);
 print_r($array2);
 print_r($array3);
?>
```

PHP Output:

```
Array
  (
     [name] => Toyota
     [type] => Celica
     [colour] => black
     [manufactured] => 1991
  )
  Array
  (
     [0] => Toyota
     [1] => Celica
     [2] => black
     [3] => 1991
  )
  Array
  (
     [name] => Toyota
     [0] => Celica
     [colour] => black
     [1] => 1991
  )
```

HTML Render:

```
Array ( [name] => Toyota [type] => Celica [colour] => black [manufactured] => 1991 ) Array
( [0] => Toyota [1] => Celica [2] => black [3] => 1991 ) Array ( [name] => Toyota [0] =>
Celica [colour] => black [1] => 1991 )
```

Example #2

The following example will output the identical text as Example #1:

```
<?php
  $array['name']="Toyota";
  $array['type']="Celica";
  $array['colour']="black";
  $array['manufactured']="1991";

  $array2[]="Toyota";
  $array2[]="Celica";
  $array2[]="black";
  $array2[]="1991";

  $array3['name']="Toyota";
```

```
  $array3[]="Celica";
  $array3['colour']="black";
  $array3[]="1991";

  print_r($array);
  print_r($array2);
  print_r($array3);
?>
```

Example #3
Using the Example #1 and Example #2 above, now you can try and use arrays the same way as normal variables:

```
 PHP Code:
<?php
 echo "Manufacturer: &lt;b&gt;{$array['name']}&lt;/b&gt;&lt;br /&gt;\n";
 echo "Brand: &lt;b&gt;{$array2['1']}&lt;/b&gt;&lt;br /&gt;\n";
 echo "Colour: &lt;b&gt;".$array3['colour']."&lt;/b&gt;&lt;br /&gt;\n";
 echo "Year Manufactured: &lt;b&gt;".$array3[1]."&lt;/b&gt;&lt;br /&gt;\n"
?>

 PHP Output:
 Manufacturer: <b>Toyota</b><br />
 Brand: <b>Celica</b><br />
 Colour: <b>black</b><br />
 Year Manufactured: <b>1991</b><br />

 HTML Render:
Manufacturer: Toyota
Brand: Celica
Colour: black
Year Manufactured: 1991
```

Multidimensional Arrays

Elements in an array can also be an array, allowing for multidimensional arrays. An example, in accordance with the motoring examples above, is:

```
<?php
$cars = array(
   "car1" => array("make" => "Toyota","colour" => "Green","year" => 1999,"engine_cc" =>
1998),
   "car2" => array("make" => "BMW","colour" => "RED","year" => 2005,"engine_cc" =>
2400),
   "car3" => array("make" => "Renault","colour" => "White","year" => 1993,"engine_cc" =>
1395),
);
?>
```

In this example, if you were to use:

```
<?php
echo "$cars['car1']['make']<br>";
echo "$cars['car3']['engine_cc']";
```

```
?>
```

The output would be:
Toyota
1395

Array Functions

There exist dozens of array manipulation functions. Before implementing your own, make sure it doesn't already exist as a PHP function in Array functions (PHP manual entry).

Array traversal

In various circumstances, you will need to visit every array element and perform a task upon it.

The simplest and the most widely used method for this is the foreach operator which loops through the whole array and works individually with each key/item couple. If a more complex way of traversing the array is needed, the following functions operate using the internal array pointer:
reset - sets the internal pointer to the first element and returns the first element
prev - sets the internal pointer to the previous element and returns it
current - returns the current element; does not change the internal pointer
next - sets the internal pointer to the next element and returns it
each - returns the current element; then sets the internal pointer to the next element
end - sets the internal pointer to the last element and returns the last element

```
<?php
// using an array's iterator to print its values in reverse order
$my_array = array('a', 'b', 'c');
end($my_array);
while($i = current($my_array)) {
        echo $i."\n";
        prev($my_array);
}
?>
```

Another possibility is defining a function and applying it to each array element via one of the following functions:
array_walk - applies a function to each array element
array_walk_recursive - same, but if the element is itself an array, it will traverse that array too

Control structures

if structure

The if Statement

Conditional structures are used to control which statements get executed. They are composed of three fundamental elements:
if statements;
elseif statements; and
else statements.

Conditionals in PHP are structured similarly to those found in C++ and Java. The structure begins with an if clause, which is composed of the word "if" followed by a true/false statement in parentheses ( ). The subsequent code will be contained in a block, denoted by curly braces { }. Sometimes the braces are omitted, and only one line will follow the if statement. elseif and else clauses sometimes occur after the if clause, to test for different statements.

The if clause says "If this statement is true, I want the program to execute the following statements. If it is false, then ignore these statements." In technical terms, it works like this: When an if statement is encountered, the true/false statement in parentheses is evaluated. If the statement is found to be true, the subsequent block of code contained in curly braces is executed. However, if the statement is found to be false, the program skips those lines and executes the next non-blank line.

Following the if clause are two optional clauses: else and elseif. The elseif clause says "If the last statement was false, let's see if this statement is true. If it is, execute the following code. If it isn't, then skip it." elseif statements are only evaluated when the preceding if statement comes out to be false. Otherwise they are skipped. Other than that, the elseif clause works just like a regular if clause. If it is true, its block is executed, if not, its block is skipped.

Finally, the else clause serves as a "catch-all" for an if statement. Essentially the else statement says "If all of the preceding tests fail, then execute this code."

Example 1
```
<?php
$foo = 1;
$bar = 2;
if($foo == $bar) {
    echo "$foo is equal to $bar.";
} elseif ($foo > $bar) {
    echo "$foo is greater than $bar.";
} else {
    echo "$foo is less than $bar.";
}
?>
```

Example 2

```
<?php
$lower = 10;
$upper = 100;
$needle = 25;
if(($needle >= $lower) && ($needle <= $upper)) {
    echo "The needle is in the haystack.";
} else if(($needle <= $lower) || ($needle >= $upper)) {
    echo "The needle is outside of the haystack.";
}
?>
```

Conditional Expressions

Conditional Values function via basic formal logic. It is important to understand how the if clause, among other clauses, evaluates these conditional values.

It is easiest to examine such with boolean values in mind, meaning that the result of a conditional value will be either TRUE or FALSE and not both. For example, if variable $x = 4, and a conditional structure is called with the expression if($x == 4), then the result of the expression will be TRUE, and the if structure will execute. However, if the expression is ($x == 0), then the result will be FALSE, and the code will not execute. This is simple enough.

This becomes more complicated when complex expressions are considered. The two basic operators that expressions can be conjoined with are the AND (&&) and OR (||).

Examples

We are given variables $x and $y.
  $x = 4;
  $y = 8;

Given the complex expression:
  ($x == 4 AND $y == 8)

We are given a result of TRUE, because the result of both separate expressions are true. When expressions are joined with the AND operator, both sides MUST be true for the whole expression to be true.
Similarly:
  ($x == 4 OR $y == 8)

We are given a result of TRUE as well, because at least one expression is true. When expressions are joined with the OR operator, at least one side MUST be true for the whole expression to be true.
Conversely,
  ($x == 4 AND $y == 10)

This expression will return FALSE, because at least one expression in the whole is false. However,
  ($x == 4 OR $y == 10)

This expression will return TRUE, because at least one expression in the whole is true.

Code Blocks

A code block is one or more statements or commands that are contained between a pair of curly braces { }. Blocks are used primarily in loops, conditionals and functions. Blocks can be nested inside one another, for instance as an if structure inside of a loop inside of a function.

If, after one of the conditional statements, there is no block of code enclosed by curly braces, only the next statement will be executed. It is recommended that you avoid using this to help prevent accidents when adding extra code after the block.
The following code will not work as intended:

```
if(FALSE)
  echo 'FALSE evaluates to true.';
  echo 'Who knew that FALSE was TRUE?';
```

The second echo statement was executed, despite the if clause. The lack of brackets caused the if statement to only apply to the first statement, making the second statement evaluate regardless of the outcome of the if statement.

To avoid this problem, make sure to use brackets with conditional statements, even if there is only a single line of code to be executed. This prevents the error in the above code from occurring when you add an extra line after the existing block.
This code fixes the previous bug.

```
if(FALSE)
{
  echo 'FALSE evaluates to true.';
  echo 'Who knew that FALSE was TRUE?';
}
```

The second echo statement should never be executed in this snippet.

Switch structure

Switch Cases

How They Work

Here's an example of a simple game where a user enters a $user_command and different functions are run as a result:

```
if($user_command == "n")
{
  go_north();
}
else if($user_command == "e")
{
  go_east();
}
else if($user_command == "s")
{
  go_south();
}
else if($user_command == "w")
{
  go_west();
}
else
{
  do_something_else();
}
```

Clearly, there's a lot of repeated code here. The switch case structure allows you to avoid this redundant code. It allows programmers to repeatedly compare the value of a certain variable to a list of possible values and execute code based on the result. This is the syntax for a switch case statement, compared to the same code written using if statements:if statement style switch case style

```
 if($firstvariable == 'comparison1'
        || $firstvariable == 'comparison2')
 {
     doSomething();
     doSomethingElse();
 }
 else if ($firstvariable == 'comparison3')
 {
     doAThirdThing();
 }
 else
 {
     launchMissiles();
 }         // Look at how much switch case saves you!
 switch($firstvariable)
```

```
{
 case 'comparison1':
 case 'comparison2':
     doSomething();
     doSomethingElse();
     break;

 case 'comparison3':
     doAThirdThing();
     break;

 default:
     launchMissiles();
     break;
}
```

The switch case style will save you from retyping $firstvariable, and make your code look cleaner (especially if that code is a long chain of simple if statements). Returning to our zork sample program, we have:Original Code Switch-Case Code

```
 if($user_command == "n")
 {
   go_north();
 }
 else if($user_command == "e")
 {
   go_east();
 }
 else if($user_command == "s")
 {
   go_south();
 }
 else if($user_command == "w")
 {
   go_west();
 }
 else
 {
   do_something_else();
 }         switch($user_command)
 {
   case 'n':
     go_north();
     break;
   case 'e':
     go_east();
     break;
   case 's':
     go_south();
     break;
   case 'w':
     go_west();
```

```
    break;
  default:
    do_something_else();
    break;
}
```

Syntax

```
switch($var)
{
 case [value]:
  [code]
  break;

 case [value]:
  [code]
  break;

 ...

 default:
  [code]
  break;
}
```

In this example. $var is the first variable to be compared. This variable is then compared against each case statement from the top down, until it finds a match. At that point, the code will excecute until a break statement is reached (which will allow you to leave the case statement entirely).

Important Warning about Using Switch Case Statements

Don't forget to use break when you mean break! If you forget, you might run functions you don't intend to. However, there are circumstances where leaving breaks out can be useful. Consider this example:

```
 switch ($n) {
  case 0:
  case 1:
  case 2:
    //only executes if $n is 0, 1 or 2
    doSomethingForNumbers2OrSmaller();
    break;
  case 3:
    //only executes if $n is 3
    doSomethingForNumber3();
  default:
    //only executes if $n is 3 or above
    doSomethingForNumbers3OrBigger();
    break;
 }
```

This kind of coding is sometimes frowned upon, since it's not always as clear to see what the code is meant to do. Also, consider commenting case statements that aren't supposed

to have a break; statement before the next case, so when others look at your code, they know not to add a break.

While loop

The Code

Example 1
```
<?php
$c = 0;
while ($c < 5) {
   echo $c++;
} ?>
```

Example 2
```
<?php
$myName="Fred";
while ($myName!="Rumpelstiltskin") {
   if ($myName=="Fred") {
      $myName="Leslie";
   }
   else {
      $myName="Rumpelstiltskin";
   }
}
echo "How did you know?\n";
?>
```

Analysis

Example 1

This is an example that prints the numbers from 0 to 4. $c starts out as 0. When the while loop is encountered, the expression $c < 5 is evaluated for truth. If it is true, then it executes what is in the curly braces. The echo statement will print 0, and then add one to $c. The program will then go back to the top of the loop and check the expression again. Since it is true again, it will then return 1 and add one to $c. It will keep doing this until $c is equal to 5, where the statement $c<5 is false. After that, it finishes.

Example 2

The first line of the program sets $myName to "Fred". After that, the while statement checks if $myName equals "Rumpelstiltskin". The != means 'does not equal', so the expression is true, and the while loop executes its code block. In the code block an if statement (see previous chapter) checks if it equals Fred. Since it does, it then reassigns $myName to equal "Leslie". Then it skips the else, since the if was true and evaluated. Then it reaches the end of the loop, so it goes back and checks if $myName does not equal "Rumpelstiltskin". Since it still doesn't, it's true, and then it goes into the loop again. This time, the if statement is false, so the else is executed. This sets $myName to "Rumplestiltskin". We again get to the end of the loop, so it goes back, and checks. Since $myName does equal "Rumpelstiltskin", the while condition is false, and it skips the loop and continues on, where it echos, "How did you know?"

New Concepts

Loops

Loops are another important basic programming technique. Loops allow programs to execute the same lines of code repeatedly, this is important for many things in programs. In PHP you often use them to layout tables in HTML and other similar functions.

Infinite Loops

Loops can be very handy, but also very dangerous! Infinite loops are loops that fail to exit, causing them to execute until the program is stopped. This is caused when no matter what occurs during the loop, the condition will never become false and therefore never exit. For example, if Example 1 subtracted 1 from $c...

```
$c=0;
while ($c<5) {
  $c--;
  echo $c;
}
```

$c will always be less than 5, no matter what, so the loop will continue forever. This causes problems for those of us who don't have infinite time (or computer memory!). So, in that case, let's learn a handy dandy little bugger.

If you add 'break;' to a loop, the loop will end, no matter whether the condition is false or not.

Parting Example

Let's combine the two examples. Before moving on take a few moments to write out the steps this program goes through and what its output is.

```
<?php
$c = 1;
$myName="Fred";
while ($myName!="Rumplestilskin") {
  if ($myName=="Fred") {
    $myName="Leslie";
  }
  else {
    $myName="Marc";
  }
  $c++;
  if ($c==3) {
    break;
  }
}
echo "You lose and I get your baby!\n";
?>
```

Do while loop.

The do while loop

The do while loop is similar in syntax and purpose to the while loop. The do/while loop construct moves the test that continues the loop to the end of the code block. The code is executed at least once, and then the condition is tested. For example:

```
<?php
$c = 6;
do {
  echo 'Hi';
} while ($c < 5);
?>
```

Even though $c is greater than 5, the script will echo "Hi" to the page one time.

PHP's do/while loop is not commonly used.

The continue statement

The continue statement causes the current iteration of the loop to end, and continues execution where the condition is checked - if this condition is true, it starts the loop again.

For loop

The FOR loop

The for loop is one of the basic looping structures in most modern programming languages. Like the while loop, for loops execute a given code block until a certain condition is met.

Syntax

The basic syntax of the for loop in PHP is similar to the c syntax:
for([initialization]; [condition]; [step])

Initialization happens the first time the loop is run. It is used to initialize variables or perform other actions that are to be performed before the first execution of the body of the loop.
The Condition is evaluated before each execution of the body of the loop; if the condition is true, the body of the loop will be executed, if it is false, the loop is exited and program execution resumes at the first line after the body of the loop.
Step specifies an action that is to be performed after each execution of the loop body.
Consider this example:

PHP Code:

```
for($i = 0; $i < 5; $i++)
{
 echo($i . "<br />");
}
```

PHP Output:

```
0<br />1<br />2<br />3<br />4<br />
```

HTML Render:

0
1
2
3
4

The loop can also be formatted without using concatenation, according to personal preference:

```
for($i = 0; $i < 5; $i++)
{
 echo "$i <br />";
}
```

Explanation

The variable $i is initialized as 0. When the loop ran once for the first time, it printed the

original value of $i, 0. Each time the loop ran, it incremented $i, then checked to see if $i was still less than 5. If it was, it continued looping. When $i finally reached 5, it was no longer less than 5, so PHP stopped looping and went to the next line after the loop code block (none, in this case).

Do note that the Initialization, Condition and Step of the For-loop can be empty. In this case, you will get an infinite loop, unless you use the "break" keyword inside the loop somwhere.

Also note that the Initialization and Step parts of the For-loop can hold more than one instruction. More info can be found via the link to the PHP manual at the end of this page.

Using FOR loops to traverse arrays

In the section on while loops the sort() example uses a while loop to print out the contents of the array. Generally programmers use for loops for this kind of job.

Example

NOTE: Use of indices like below is highly discouraged. Use the key-value for-loop construct.

```
 $menu = array("Toast and Jam", "Bacon and Eggs", "Homefries", "Skillet", "Milk and Cereal");
 // note to self: get breakfast after writing this article
 $count = count($menu);
 for($i = 0; $i < $count; $i++)
 {
  echo($i + 1 . ". " . $menu[$i] . "<br />");
 }
```

Again, this can be formatted without concatenation if you prefer:

```
 for($i = 0; $i < $count; $i++)
 {
  $j=$i+1;
  echo "$j. {$menu[$i]} <br />";
 }
```

Explanation

```
 $count = count($menu);
```

We define the count before the for loop for more efficient processing. This is because each time the for loop is run (whilst $i < $count) it evaluates both sides of the equation and executes any functions. If we put $i < count($menu), This would evaluate count($menu) each time the process is executed which is costly when dealing with large arrays.

```
 for($i = 0; $i < $count; $i++)
```

This line sets up the loop. It initializes the counter, $i, to 0 at the start, adds one every time the loop goes through, and checks that $i is less than the size of the array.

```
 {
  echo($i + 1 . ". " . $menu[$i] . "<br />");
 }
```

The echo statement is pretty self-explanatory, except perhaps the bit at the start, where we echo $i + 1. We do that because, as you may recall, arrays start at 0 and end at n - 1 (where n is their length), so to get a numbered list starting at one, we have to add one to the counter each time we print it.

Of course, as I mentioned before, both pieces of code produce the following output:
1. Toast and Jam
2. Bacon and Eggs
3. Homefries
4. Skillet
5. Milk and Cereal

Believe it or not, there's actually a way to traverse arrays that requires even less typing. (And isn't that the goal?) Check out the FOREACH loop for another way of doing what we did here.

Foreach loop

The Code

```
foreach ($array as $someVar) {
  echo ($someVar . "<br />");
}
```

or:

```
foreach ($array as $key => $someVar) {
  echo ($key."holds the value ".$someVar."<br />");
}
```

Analysis

The foreach loop is a special form of the standard for loop. The example above will print all the values of $array. The foreach structure is a convenient way to loop through an array.

simple foreach statement

Foreach loops are useful when dealing with an array indexed with arbitrary keys (e.g. non-numeric ones):

```
$array = array(
  "1st" => "My House",
  "2nd" => "My Car",
  "3rd" => "My Lab"
);
```

To use the classical for structure, you'd have to write:

```
// get all the array keys
$arrayKeys = array_keys($array);
// loop through the keys
for ($i=0; $i<count($array); $i++) {
  // get each array value using its key
  echo $array[($arrayKeys[$i])] . "<br />";
}
```

Basically, an array value can be accessed only from its key: to make sure you get all the values, you first have to make a list of all the existings keys then grab all the corresponding values. The access to first aray value, the previous example does the following steps:

```
$firstKey = $arrayKeys[0]; // which is '1st'
$firstValue = $array[$firstKey]; // which is 'My House' ($array('1st'))
```

The foreach structure does all the groundwork for you:

```
foreach ($array as $someVar) {
   echo $someVar . "<br />";
}
```

Note how the latter example is easier to read (and write). Both will output:
My House
My Car
My Lab

foreach with key values

If you need to use the array keys in your loop, just add the variable as in the following statement:

```
foreach ($array as $myKey => $value) {
  // use $myKey
}
```

Note that this is very usefull when constructing a dropdown list in HTML. You can use a foreach-loop to have the $myKey variable inserted into the value="..." part and the $value as the actuall text.
This form mimics the way we used custom keys for $array elements. It will not only assign the elements of $array to $someVar, but also assign the keys of those elements to $i.

PHP Code:

```
<?php
$array = array("1st" => "My House", "2nd" => "My Car", "3rd" => "My Lab");
foreach ($array as $i => $someVar) {
  echo $i . ": " . $someVar . "<br />\n";
}
?>
```

PHP Output:

```
1st: My House<br />
2nd: My Car<br />
3rd: My Lab<br />
```

HTML Render:

1st: My House
2nd: My Car
3rd: My Lab

Note that if you change the assigned variable inside the foreach loop, the change will not be reflected to the array. Therefore if you need to change elements of the array you need to change them by using the array key.

Example:

```
$array = array(
  "1st" => "My House",
  "2nd" => "My Car",
```

```
  "3rd" => "My Lab"
);

foreach ($array as $i => $someVar) {

  // OK
  if($someVar == 'My Lab')
    $array[$i] = 'My Laboratory';

  // doesn't update the array
  $someVar = 'Foo';
}
```

Functions

Introduction

Functions (often called methods) are a way to group common tasks or calculations to be re-used easily.

Functions in computer programming are much like mathematical functions: You can give the function values to work with and get a result without having to do any calculations yourself.

You can also find a huge list of predefined functions built into PHP in the PHP Manual's function reference.

How to call a function

Note that echo is not a function.[1] "Calling a function" means causing a particular function to run at a particular point in the script. The basic ways to call a function include:
calling the function to write on a new line (such as after a ";" or "}")
print('I am Naresh, I am.');
calling the function to write on a new line inside a control

```
<?php
if ($a==72){
  print('I am Naresh, I am.');
}
?>
```

assigning the returned value of a function to a variable "$var = function()"

```
<?php
$result = sum ($a, 5);
?>
```

calling a function inside the argument parentheses (expression) of a control

```
<?php
while ($i < count($one)){

}
?>
```

In our earlier examples we have called several functions. Most commonly we have called the function print() to print text to the output. The parameter for echo has been the string we wanted printed (for example print("Hello World!") prints "Hello World!" to the output).

If the function returns some information we assign it to a variable with a simple =:
$var1 = func_name();
[edit]

Parameters

Parameters are variables that exist only within that function. They are provided by the programmer when the function is called and the function can read and change them locally (except for reference type variables, which are changed globally - this is a more advanced topic).

When declaring or calling a function that has more than one parameter, you need to separate between different parameters with a comma ','.

A function declaration can look like this:

```
function print_two_strings($var1, $var2)
{
  echo $var1;
  echo "\n";
  echo $var2;
  return NULL;
}
```

To call this function you must give the parameters a value. It doesn't matter what the value is, as long as there is one.

function call:

```
print_two_strings("hello", "world");
```

Output:
hello
world

When declaring a function you sometimes want to have the freedom not to use all the parameters, therefore PHP allows you to give them default values when declaring the function:

```
function print_two_strings($var1 = "Hello World", $var2 = "I'm Learning PHP")
{
  echo($var1);
  echo("\n");
  echo($var2);
}
```

These values will only be used if the function call does not include enough parameters. If there is only one parameter provided then $var2 = "I'm Learning PHP".

function call:

```
print_two_strings("hello");
```

Output:
hello
I'm Learning PHP

Another way to have a dynamic number of parameters is to use PHP's built-in func_num_args, func_get_args, and func_get_arg functions.

```
function mean()
{
  $sum = 0;
  $param_count = func_num_args();
  for ($i = 0; $i < $param_count; $i++)
  {
    $sum += func_get_arg($i);
  }
  $mean = $sum / $param_count;
  echo "Mean: {$mean}";
  return NULL;
}
```

Or:

```
function mean()
{
  $sum = 0;
  $vars = func_get_args();
  for ($i = 0; $i < count($vars); $i++)
  {
    $sum += $vars[$i];
  }
  $mean = $sum / count($vars);
  echo "Mean: {$mean}";
  return NULL;
}
```

The above functions would calculate the arithmetic mean of all of the values passed to them and output it. The difference is that the first function uses func_num_args and func_get_arg, while the second uses func_get_args to load the parameters into an array. The output for both of them would be the same. For example:

```
mean(35, 43, 3);
```

Output:
Mean: 27

==Returning a value==test

This function is all well and good, but usually you will want your function to return some information. Generally there are 2 reasons why a programmer would want information from a function:
The function does tasks such as calculations, and we need the result.
A function can return a value to indicate if the function encountered any errors.

To return a value from a function use the return() statement in the function.

```
function add_numbers($var1 = 0, $var2 = 0, $var3 = 0)
{
  $var4 = $var1 + $var2 + $var3;
```

```
  return $var4;
}
```

Example PHP script:

```
<?php
function add_numbers($var1 = 0, $var2 = 0, $var3 = 0)
{
  $var4 = $var1 + $var2 + $var3;
  return $var4;
}

$sum = add_numbers(1,6,9);
echo "The result of 1 + 6 + 9 is {$sum}";
?>
```

Result:
The result of 1 + 6 + 9 is 16

Notice that a return() statement ends the function's course. If anything appears in a function declaration after the return() statement is executed, it is parsed but not executed. This can come in handy in some cases. For example:

```
<?php
function divide ($dividee, $divider) {
  if ($divider == 0) {
    //Can't divide by 0.
    return false;
  }
  $result = $dividee/$divider;
  return $result;
}
?>
```

Notice that there is no else after the if. This is due to the fact that if $divider does equal 0, the return() statement is executed and the function stops.

If you want to return multiple variables you need to return an array rather than a single variable. For example:

```
<?php
function maths ($input1, $input2) {
  $total = ($input1 + $input2);
  $difference = ($input1 - $input2);
  $ret = array("tot"=>$total, "diff"=>$difference);
  return $ret;
}
?>
```

When calling this from your script you need to call it into an array. For example:

```
<?php
  $return=maths(10, 5);
?>
```

In this case $return['tot'] will be the total (eg 15), while $return['diff'] will be the difference (5).

Runtime function usage

A developer can create functions inside a PHP script without having to use the function name($param...) {} syntax. This can done by way of programming that can let you run functions dynamically.

Executing a function that is based on a variable's name

There are two ways to do it, either using direct call or call_user_func or call_user_func_array:

Using call_user_func* functions to call functions

call_user_func and call_user_func_array only differ that the call_user_func_array allows you to use the second parameter as array to pass the data very easily, and call_user_func has an infinite number of parameters that is not very useful in a professional way. In these examples, a class will be used for a wider range of using the example:

```
<?php
class Some_Class {
  function my_function($text1,$text2,$text3) {
    $return = $text1."\n\n".$text2."\n\n".$text3;
    return $return;
  }
}
$my_class=new Some_Class();
?>
```

Using call_user_func:

```
<?php
$one = "One";
$two = "Two";
$three = "Three";
$callback_func = array(&$my_class,"my_function");
$result = call_user_func($callback_func,$one,$two,$three);
echo $result;
?>
```

Using call_user_func_array:

```
<?php
$one = "One";
$two = "Two";
$three = "Three";
```

```
$callback_func = array(&$my_class,"my_function");
$result = call_user_func_array($callback_func,array($one,$two,$three));
echo $result;
?>
```

Note how call_user_func and call_user_func_array are used in both of the examples. call_user_func_array allows the script to execute the function more dynamically.

As there was no example of using both of these functions for a non-class function, here they are: Using call_user_func:

```
<?php
$one = "One";
$two = "Two";
$three = "Three";
$callback_func = "my_function";
$result = call_user_func($callback_func,$one,$two,$three);
echo $result;
?>
```

Using call_user_func_array:

```
<?php
$one = "One";
$two = "Two";
$three = "Three";
$callback_func = "my_function";
$result = call_user_func_array($callback_func,array($one,$two,$three));
echo $result;
?>
```

More complicated examples

```
<?php
$my_func($param1, $param2);
$my_class_name = new ClassObject();
$my_class_name->$my_func_from_that_class($param1, $param2);
// The -> symbol is a minus sign follow by a "larger then" sign. It allows you to use a function that is defined in a different PHP class.
// It comes directly from Object-Oriented programming. Via a constructor, a function of that class is executable.
// This specific example is a function that returns no values.
call_user_func($my_func, $param1, $param2);

call_user_func(array(&${$my_class_name}, $my_func), $param1, $param2);

call_user_func_array($my_func, array($param1, $param2));
// Most powerful, dynamic example
call_user_func_array(array(&${$my_class_name}, $my_func), array($param1, $param2));
?>
<?php
 function positif ($x + $y;) {
```

```
 $x=2;
 $y=5;
 $z = $x+$y;
echo $z;
 }
positif=$x+$y;
?>
```

TODO: Somebody add what the & in codeline 5 means

Creating runtime functions

Creating runtime functions is a very good way of making the script more dynamic:

```
<?php
$function_name=create_function('$one, $two','return $one+$two;');
echo $function_name."\n\n";
echo $function_name("1.5", "2");
?>
```

create_function creates a function with parameters $one and $two, with a code to evaluate return... When create_function is executed, it stores the function's info in the memory and returns the function's name. This means that you cannot customise the name of the function although that would be preferred by most developers.

Files

Working with files is an important part of any programming language and PHP is no different. Whatever your reasons are for wanting to manipulate files, PHP will happily accommodate them through the use of a handful of functions. You should have read and be comfortable with the concepts presented in the first five sections of this book before beginning here.

Fopen() and Fclose()

Fopen() is the basis for file manipulation. It opens a file in a certain mode (which you specify) and returns a handle. Using this handle you can read and/or write to the file, before closing it with the fclose() function.

Example Usage

```
<?php
   $Handle = fopen('data.txt', 'r'); // Open the file for reading
   fclose($Handle); // Close the file
?>
```

In the example above you can see the file is opened for reading by specifying 'r' as the mode. For a full list of all the modes available to fopen() you can look at the PHP Manual page.

Opening and closing the file is all well and good but to perform useful operations you need to know about fread() and fwrite().

When a PHP script finishes executing, all open files are automatically closed. So although it is not strictly necessary to close a file after opening it, it is considered good programming practice to do so.

Reading

Reading can be done a number of ways. If you just want all the contents of a file available to work with you can use the file_get_contents() function. If you want each line of the file in an array you can use the file() command. For total control over reading from files fread() can be used.

These functions are usually interchangeable and each can be used to perform each other's function. The first two do not require that you first open the file with fopen() or then close it with fclose(). These are good for quick, one-time file operations. If you plan on performing multiple operations on a file it is best to use fopen() in conjunction with fread(), fwrite() and fclose() as it is more efficient.
An example of using file_get_contents()

Code:

```
<?php
    $Contents = file_get_contents('data.txt');
    echo $Contents;
?>
```

Output:

```
I am the contents of data.txt
```

This function reads the entire file into a string and from then on you can manipulate it as you would any string.

An example of using file()

Code:

```
<?php
    $Lines = file('data.txt');
    foreach($Lines as $Key => $Line) {
        $LineNum = $Key + 1;
        echo "Line $LineNum: $Line";
    }
?>
```

Output:

```
Line 1: I am the first line of file
Line 2: I am the second line the of the file
Line 3: If I said I was the fourth line of the file, I'd be lying
```

This function reads the entire file into an array. Each item in the array corresponds to a line in the file.

An example of using fread()

Code:

```
<?php
    $Handle = fopen('data.txt', 'r');
    $String = fread($Handle, 64);
    fclose($Handle);

    echo $String;
?>
```

Output:

```
I am the first 64 bytes of data.txt (if it was ASCII encoded). I
```

This function can read up to the specified number of bytes from the file and return it as a string. For the most part, the first two functions will be preferable but there are occasions when this function is needed.

As you can see, with these three functions you are able to easily read data from a file into a form that is convenient to work with. The next part shows how these functions can be used to do the jobs of the others but this is optional. You may skip it and move onto the

Writing section if you are not interested.
.

```
<?php
   $File = 'data.txt';

   function DetectLineEndings($Contents) {
      if(false !== strpos($Contents, "\r\n")) return "\r\n";
      elseif(false !== strpos($Contents, "\r")) return "\r";
      else return "\n";
   }

   /* This is equivalent to file_get_contents($File) but is less efficient */
   $Handle = fopen($File, 'r');
   $Contents = fread($Handle, filesize($File));
   fclose($Handle);

   /* This is equivalent to file($File) but requires you to check for the line-ending
   type. Windows systems use \r\n, Macintosh \r and Unix \n. File($File) will
   automatically detect line-endings whereas fread/file_get_contents won't */
   $LineEnding = DetectLineEndings($Contents);
   $Contents = file_get_contents($File);
   $Lines = explode($LineEnding, $Contents);

   /* This is also equivalent to file_get_contents($File) */
   $Lines = file($File);
   $Contents = implode("\n", $Lines);

   /* This is equivalent to fread($File, 64) if the file is ASCII encoded */
   $Contents = file_get_contents($File);
   $String = substr($Contents, 0, 64);
?>
```

Writing

Writing to a file is done by using the fwrite() function in conjunction with fopen() and fclose(). As you can see, there aren't as many options for writing to a file as there are for reading from one. However, PHP 5 introduces the function file_put_contents() which simplifies the writing process somewhat. This function will be discussed later in the PHP 5 section as it is fairly self-explanatory and does not require discussion here.

The extra options for writing don't come from the amount of functions, but from the modes available for opening the file. There are three different modes you can supply to the fopen() function if you wish to write to a file. One mode, 'w', wipes the entire contents of the file so anything you then write to the file will fully replace what was there before. The second mode, 'a' appends stuff to the file so anything you write to the file will appear just after the original contents of the file. The final mode 'x' only works for non-existent files. All three writing modes will attempt to create the file if it doesn't exist whereas the 'r' mode will not.
An example of using the 'w' mode

Code:
```
<?php
    $Handle = fopen('data.txt', 'w'); // Open the file and delete its contents
    $Data = "I am new content\nspread across\nseveral lines.";
    fwrite($Handle, $Data);
    fclose($Handle);

    echo file_get_contents('data.txt');
?>
```

Output:
```
I am new content
spread across
several lines.
```

An example of using the 'a' mode

Code:
```
<?php
    $Handle = fopen('data.txt', 'a'); // Open the file for appending
    $Data = "\n\nI am new content.";
    fwrite($Handle, $Data);
    fclose($Handle);

    echo file_get_contents('data.txt');
?>
```

Output:
```
I am the original content.

I am new content.
```

An example of using the 'x' mode

Code:
```
<?php
    $Handle = fopen('newfile.txt', 'x'); // Open the file only if it doesn't exist
    $Data = "I am this file's first ever content!";
    fwrite($Handle, $Data);
    fclose($Handle);

    echo file_get_contents('newfile.txt');
?>
```

Output:
```
I am this file's first ever content!
```

Of the three modes shown above, 'w' and 'a' are used the most but the writing process is essentially the same for all the modes.

Reading and Writing

If you want to use fopen() to open a file for both reading and writing all you need to do is put a '+' on the end of the mode. For example, reading from a file requires the 'r' mode. If you want to read and write to/from that file you need to use 'r+' as a mode. Similarly you can read and write to/from a file using the 'w+' mode however this will also truncate the file to zero length. For a better description visit the fopen() page which has a very useful table describing all the modes available.

Error Checking

Error checking is important for any sort of programming but when working with files in PHP it is especially important. This need for error checking arises mainly from the filesystem the files are on. The majority of webservers today are Unix-based and so, if you are using PHP to develop web-applications, you have to account for file permissions. In some cases PHP may not have permission to read the file and so if you've written code to read a particular file, it will result in an ugly error. More likely is that PHP doesn't have permission to write to a file and that will again result in ugly errors. Also, the file's existence is (somewhat obviously) important. When attempting to read a file, you must make sure the file exists first. On the other side of that, if you're attempting to create and then write to a file using the 'x' mode then you must make sure the file doesn't exist first.

In short, when writing code to work with files, always assume the worst. Assume the file doesn't exist and you don't have permission to read/write to it. In most cases this means you have to tell the user that, in order for the script to work, he/she needs to adjust those file permissions so that PHP can create files and read/write to them but it also means that your script can adjust and perform an alternative operation.

There are two main ways of error checking. The first is by using the '@' operator to suppress any errors when working with the file and then checking if the result is false or not. The second method involves using more functions like file_exists(), is_readable() and is_writeable().
Examples of using the '@' operator

```
<?php
   $Handle = @ fopen('data.txt', 'r');
   if(!$Handle) {
      echo 'PHP does not have permission to read this file or the file
in question doesn\'t exist.';
   } else {
      $String = fread($Handle, 64);
      fclose($Handle);
   }

   $Handle = @ fopen('data.txt', 'w'); // The same applies for 'a'
   if(!$Handle) {
      echo 'PHP either does not have permission to write to this file or
it does not have permission to create this file in the current directory.';
   } else {
      fwrite($Handle, 'I can has content?');
```

```
        fclose($Handle);
    }

    $Handle = @ fopen('data.txt', 'x');
    if(!$Handle) {
        echo 'Either this file exists or PHP does not have permission to
create this file in the current directory.';
    } else {
        fwrite($Handle, 'I can has content?');
        fclose($Handle);
    }
?>
```

As you can see, the '@' operator is used mainly when working with the fopen() function. It can be used in other cases but is generally less efficient.

Examples of using specific checking functions

```
<?php
    $File = 'data.txt';

    if(!file_exists($File)) {
        // No point in reading since there is no content
        $Contents = '';

        // But might want to create the file instead
        $Handle = @ fopen($File, 'x'); // Still need to error-check ;)
        if(!$Handle) {
            echo 'PHP does not have permission to create a file in the current directory.';
        } else {
            fwrite($Handle, 'Default data');
            fclose($Handle);
        }
    } else {
        // The file does exist so we can try to read its contents
        if(is_readable($File)) {
            $Contents = file_get_contents($File);
        } else {
            echo 'PHP does not have permission to read that file.';
        }
    }

    if(file_exists($File) && is_writeable($File)) {
        $Handle = fopen($File, 'w');
        fwrite($Handle, 'I can has content?');
        fclose($Handle);
    }
?>
```

You can see by that last example that error-checking makes your code very robust. It allows it to be prepared for most situations and behave accordingly which is an essential

aspect of any program or script.

Line-endings

Line-endings were mentioned briefly in the final example in the 'Reading' section of this chapter and it is important to be aware of them when working with files. When reading from a text file it is important to know what types of line-endings that file contains. 'Line-endings' are special characters that try to tell a program to display a new line. For example, Notepad will only move a piece of text to a new line if it finds "\r\n" just before the new line (it will also display new lines if you put word wrap on).

If someone writes a text file on a Windows system, the chances are that each line will end with "\r\n". Similarly, if they write the file on a Classic Macintosh (Mac OS 9 and under) system, each line will probably end with "\r". Finally, if they write the file on a Unix-based system (Mac OS X and GNU/Linux), each line will probably end with "\n".

Why is this important? Well, when you read a file into a string with file_get_contents(), the string will be one long line with those line-endings all over the place. Sometimes they will get in the way of things you want to do with the string so you can remove them with:

```
<?php
   $String = str_replace(array("\n", "\r"), '', $String);
?>
```

Other times you may need to know what kind of line-ending is being used throughout the text in order to be consistent with any new text you add. Luckily, in 99% of cases, the line-endings will never change type throughout the text so the custom function 'DetectLineEndings' can be used as a quick way of checking:

```
<?php
   function DetectLineEndings($String) {
      if(false !== strpos($String, "\r\n")) return "\r\n";
      elseif(false !== strpos($String, "\r")) return "\r";
      else return "\n";
   }
?>
```

Most of the time though, it is just sufficient to be aware of their existence within the text so you can adjust your script to cope properly.

Binary Safe

So far, all of the text seen in this chapter has been assumed to be encoded in some form of plaintext encoding such as UTF-8 or ASCII. Files do not have to be in this format however and in fact there exist a huge number of formats that are aren't (such as pictures or executables). If you want to work with these files you have to ensure that the functions you are using are 'binary safe'. Previously you would have to add 'b' to the end of the modes you used to tell PHP to treat the file as a binary file. Failing to do so would give unexpected results and generally 'weird-looking' data.

Since about PHP 4.3, this is no longer necessary as PHP will automatically detect if it needs to open the file as a text file or a binary file and so you can still follow most of the examples shown here.

Working with binary data is a lot different to working with plaintext strings and characters and involves many more functions which are beyond the scope of this chapter however it is important you know about these differences.

Serialization

Serialization is a technique used by programmers to preserve their working data in a format that can later be restored to its previous form. In simple cases this means converting a normal variable such as an array into a string and then storing it somewhere. That data can then be unserialized and the programmer will be able to work with the array once again.

There is a whole chapter devoted to Serialization in this book as it is a useful technique to know how to use effectively. It is mentioned here as one of the primary uses of serialization is to store data on plain files when a database is not available. It is also used to store the state of a script and to cache data for quicker access later and files are one of the preferred media for this storage.

In PHP, serialization is very easy to perform through use of the serialize() and unserialize() functions. Here follows an example of serialization used in conjunction with file functions. An example of storing user details in a file so that they can be easily retrieved later.

Code:

```
<?php
    /* This part of the script saves the data to a file */
    $Data = array(
        'id' => 114,
        'first name' => 'Foo',
        'last name' => 'Bartholomew',
        'age' => 21,
        'country' => 'England'
    );
    $String = serialize($Data);

    $Handle = fopen('data.dat', 'w');
    fwrite($Handle, $String);
    fclose($Handle);

    /* Then, later on, we retrieve the data from the file and output it */
    $String = file_get_contents('data.dat');
    $Data = unserialize($String);

    $Output = '';
    foreach($Data as $Key => $Datum) {
        $Field = ucwords($Key);
        $Output .= "$Field: $Datum\n";
    }
   echo $Output
?>
```

Output:

Id: 114
First Name: Foo
Last Name: Bartholomew
Age: 21
Country: England
PHP 5

There is one particular function specific to files that was introduced in PHP 5. That was the file_put_contents() function. It offers an alternative method of writing to files that does not exist in PHP 4. To see how it differs, it is easiest to just look at an example.
Examples showing writing to a file in PHP 4 and the equivalent in PHP 5 with the file_put_contents() function

```
<?php
    $File = 'data.txt';
    $Content = 'New content.';

    // PHP 4, overwrite entire file with data
    $Handle = fopen($File, 'w');
    fwrite($Handle, $Content);
    fclose($Handle);

    // PHP 5
    file_put_contents($File, $Content);

    // PHP 4, append to a file
    $Handle = fopen($File, 'a');
    fwrite($Handle, $Content);
    fclose($Handle);

    // PHP 5
    file_put_contents($File, $Content, FILE_APPEND);
?>
```

File_put_contents() will also attempt to create the file if it doesn't exist and it is binary safe. There is no equivalent of the 'x' mode for file_get_contents().

File_put_contents() is almost always preferable over the fopen() method except when performing multiple operations on the same file. It is more preferable to use it for writing than file_get_contents() is for reading and for this reason, a function is provided here to emulate the behaviour of file_put_contents() for PHP 4:

```
<?php
    if(!function_exists('file_put_contents')) {
        function file_put_contents($File, $Data, $Append = false) {
            if(!$Append) $Mode = 'w';
            else $Mode = 'a';

            $Handle = @ fopen($File, $Mode);
            if(!$Handle) return false;

            $Bytes = fwrite($Handle, $Data);
            fclose($Handle);
```

```
            return $Bytes;
        }
    }
?>
```

Mail

The mail function is used to send E-mail Messages through the SMTP server specified in the php.ini Configuration file.
bool mail (string to, string subject, string message [, string additional_headers [, string additional_parameters]])

The returned boolean will show whether the E-mail has been sent successfully or not.

This example will send message "message" with the subject "subject" to email address

"example@domain.tld". Also, the receiver will see that the eMail was sent from "Example2 <example2@domain.tld>" and the receiver should reply to "Example3 <example3@domain.tld>"

```
<?php
mail(
 "example@domain.tld", // E-Mail address
 "subject", // Subject
 "message", // Message
 "From: Example2 <example2@domain.tld>\r\nReply-to: Example3
<example3@domain.tld>) // Additional Headers
;
?>
```

There is no requirement to write E-mail addresses in format "Name <email>", you can just write "email".

This will send the same message as the first example but includes From: and Reply-To: headers in the message. This is required if you want the person you sent the E-mail to be able to reply to you. Also, some E-mail providers will assume mail is spam if certain headers are missing so unless you include the correct headers your mail will end up in the junk mail folder.

Important notes

PHP by default does not have any mail sending ability itself. It needs to pass the mail to a local mail transfer agent, such as sendmail. This means you cannot just run PHP by itself and expect it to send mail; you must have a mail transfer agent installed.
Make sure you do not have any newline characters in the to or subject, or the mail may not be sent properly.
However, the additional headers field -- which should always include the From: header -- may also include other headers. On PHP for Windows, each header should be followed by \r\n but on Unix versions, you should only include \r between header lines. Don't put \n or \r\n after the final additional header line.
The to parameter should not be an address in the form of "Name <someone@example.com>". The mail command may not parse this properly while talking with the MTA (Particularly under Windows).

Error Detection

Especially when sending multiple emails, such as for a newsletter script, error detection is important.

Use this script to send mail while warning for errors:

```
$result=@mail($to,$subject,$message,$headers);
if ( $result ) echo "Email sent successfully."
else echo "Email was not sent, as an error occurred."
```

Sending To Multiple Addresses Using Arrays

In the case below the script has already got a list of emails, we simply use the same procedure for using a loop in PHP with mysql results. The script below will attempt to send an email to every single email address in the array until it runs out.

```
while ($row = mysql_fetch_assoc($result)) {
    mail($row['email'], $subject, $message, null,"-f$fromaddr");
}
```

Then if we integrate the error checking into the multiple email script we get the following

```
$errors = 0
$sent = 0

while ($row = mysql_fetch_assoc($result)) {
    $result = "";
    $result = @mail($row['email'], $subject, $message, null,"-f$fromaddr");
    if ( !$result ) $errors = $errors + 1;
    $sent = $sent + 1;
}

echo "You have sent $sent messages";
echo "However there were $errors messages";
```

Cookies

Cookies are small pieces of data stored as text on the client's computer. Normally cookies are used only to store small amounts of data. Even though cookies are not harmful some people do not permit cookies due to concerns about their privacy. In this case you have to use Sessions.

This lesson covers setting and retrieving data from cookies.

Setting a cookie

Setting a cookie is extremely easy with setcookie().
setcookie("test", "PHP-Hypertext-Preprocessor", time()+60, "/location",1);

Here the setcookie function is being called with four arguments (setcookie has 1 more optional argument, not used here). In the above code, the first argument is the cookie name, the second argument is the cookie contents and the third argument is the time after which the cookie should expire in seconds (time() returns current time in seconds, there time()+60 is one minute from now). The path, or location, element may be omitted, but it does allow you to easily set cookies for all pages within a directory, although using this is not generally recommended.

You should note that since cookies are sent with the HTTP headers the code has to be at the top of the page (Yes, even above the DOCTYPE declaration). Any other place will generate an error.

Retrieving cookie data

If a server has set a cookie the browser sends it to the server each time a page loads. The name of each cookie sent by your server is stored in the superglobal array _COOKIE. So in the above example the cookie would be retrieved by calling $_COOKIE['test']. To access data in the cookie we use explode(). explode() turns a string into an array with a certain delimiter present in the string. That is why we used those dashes(- hyphens) in the cookie contents. So to retrieve and print out the full form of PHP from the cookie we use the code:

```
<?php
$array = explode("-", $_COOKIE['test']); //retrieve contents of cookie
print("PHP stands for ".$array[0].$array[1].$array[2]); //display the content
?>
```

Note: $_COOKIE was Introduced in 4.1.0. In earlier versions, use $HTTP_COOKIE_VARS.

Where are cookies used?

Cookies can be often used for:
user preferences

inventories
quiz or poll results
shopping carts
user authentication
remembering data over a longer period

You should never store unencrypted passwords in cookies as cookies can be easily read by the users.

You should never store critical data in cookies as cookies can be easily removed or modified by users.

Sessions

Sessions allow the PHP script to store data on the web server that can be later used, even between requests to different php pages. Every session has got a different identifier, which is sent to the client's browser as a cookie or as a $_GET variable. Sessions end when the user closes the browser, or when the web server deletes the session information, or when the programmer explicitly destroys the session. In PHP it's usually called PHPSESSID. Sessions are very useful to protect the data that the user wouldn't be able to read or write, especially when the PHP developer doesn't want to give out information in the cookies as they are easily readable. Sessions can be controlled by the $_SESSION superglobal. Data stored in this array is persistent throughout the session. It is a simple array. Sessions are much easier to use than cookies, which helps PHP developers a lot. Mostly, sessions are used for user logins, shopping carts and other additions needed to keep browsing smooth. PHP script can easily control the session's cookie which is being sent and control the whole session data. Sessions are always stored in a unique filename, either in a temporary folder or in a specific folder, if a script instructs to do so.

Using Sessions

At the top of each php script that will be part of the current session there must be the function session_start(). It must be before the first output (echo or others) or it will result in an error "Headers already sent out".
session_start();

This function will do these actions:
It will check the _COOKIE or _GET data, if it is given
If the session file doesn't exist in the session.save_path location, it will :
Generate a new Unique Identifier, and
Create a new file based on that Identifier, and
Send a cookie to the client's browser
If it does exist, the PHP script will attempt to store the file's data into _SESSION variable for further use

Now, you can simply set variables in 2 different ways, the default method:
$_SESSION['example'] = "Test";

Or the deprecated method:
$example="Test";
session_register($example);

Both of the above statements will register the session variable $_SESSION['example'] as "Test". The deprecated method should not be used, it is only listed because you can still see it in scripts written by programmers that don't know the new one. The default method is preferred.

Session Configuration Options

PHP Sessions are easy to control and can be made even more secure or less secure with small factors. Here are runtime options that can be easily changed using php_ini()

function:Name Default Changeable
session.save_path "/tmp" PHP_INI_ALL
session.name "PHPSESSID" PHP_INI_ALL
session.save_handler "files" PHP_INI_ALL
session.auto_start "0" PHP_INI_ALL
session.gc_probability "1" PHP_INI_ALL
session.gc_divisor "100" PHP_INI_ALL
session.gc_maxlifetime "1440" PHP_INI_ALL
session.serialize_handler "php" PHP_INI_ALL
session.cookie_lifetime "0" PHP_INI_ALL
session.cookie_path "/" PHP_INI_ALL
session.cookie_domain "" PHP_INI_ALL
session.cookie_secure "" PHP_INI_ALL
session.use_cookies "1" PHP_INI_ALL
session.use_only_cookies "0" PHP_INI_ALL
session.referer_check "" PHP_INI_ALL
session.entropy_file "" PHP_INI_ALL
session.entropy_length "0" PHP_INI_ALL
session.cache_limiter "nocache" PHP_INI_ALL
session.cache_expire "180" PHP_INI_ALL
session.use_trans_sid "0" PHP_INI_SYSTEM/PHP_INI_PERDIR
session.bug_compat_42 "1" PHP_INI_ALL
session.bug_compat_warn "1" PHP_INI_ALL
session.hash_function "0" PHP_INI_ALL
session.hash_bits_per_character "4" PHP_INI_ALL
url_rewriter.tags "a=href,area=href,frame=src,input=src,form=fakeentry"
PHP_INI_ALL

A simple example of this use would be this code:
//Setting The Session Saving path to "sessions", must be protected from reading
session_save_path("sessions"); // This function is an alternative to
ini_set("session.save_path","sessions");
//Session Cookie's Lifetime (not effective, but use!)
ini_set("session.cookie_lifetime",time()+60*60*24*500);
//Change the Session Name from PHPSESSID to SessionID
session_name("SessionID");
//Start The session
session_start();
//Set a session cookie (Required for some browsers, as settings that had been done before are not very effective
setcookie(session_name(), session_id(), time()+3600*24*365, "/");
This example simply sets the cookie for the next year.

Ending a Session

When user clicks "Logout", or "Sign Off", you would usually want to destroy all the login data so nobody could have have access to it anymore. The Session File will be simply deleted as well as the cookie to be unset by:
session_destroy();

Using Session Data of Other Types

Simple data such as integers, strings, and arrays can easily be stored in the $_SESSION superglobal array and be passed from page to page. But problems occur when trying to store the state of an object by assignment. Object state can be stored in a session by using the serialize() function. serialize() will write the objects data into an array which then can be stored in a $_SESSION superglobal. unserialize() can be used to restore the state of an object before trying to access the object in a page that is part of the current session. If objects are to be used across multiple page accesses during a session, the object definition must be defined before calling unserialize(). Other issues may arise when serializing and unserializing objects.

Avoiding Session Fixation
Wikipedia has related information at
Session fixation

Session fixation describes an attack vector in which a malicious third-party sets (i.e. fixes) the session identifier (SID) of a user, and is thus able to access that user's session. In the base-level implementation of sessions, as described above, this is a very real vulnerability, and every PHP program that uses sessions for anything at all sensitive should take steps to remedy it. The following, in order of how widely applicable they are, are the measures to take to prevent session fixation:

Do not use GET or POST variables to store the session ID (under most PHP configurations, a cookie is used to store the SID, and so the programmer doesn't need to do anything to implement this);
Regenerate the SID on each user request (using session_regenerate_id() at the beginning of the session);
Use session time-outs: for each user request, store the current timestamp, and on the next request check that the timeout interval has not passed;
Provide a logout function;

Check the 'browser fingerprint' on each request. This is a hash, stored in a $_SESSION variable, comprising some combination of the user-agent header, client IP address, a salt value, and/or other information. See below for more discussion of the details of this; it is thought by some to be nothing more than 'security through obscurity'. [TODO]
Check referrer: this does not work for all systems, but if you know that users of your site must be coming from some known domain you can discard sessions tied to users from elsewhere. Relies on the user agent providing the Referrer header, which should not be assumed.

This example code addresses all of the above points save the referrer check.

```
$timeout = 3 * 60; // 3 minutes
$fingerprint = md5('SECRET-SALT'.$_SERVER['HTTP_USER_AGENT']);
session_start();
if ( (isset($_SESSION['last_active']) && (time() > ($_SESSION['last_active']+$timeout)))
    || (isset($_SESSION['fingerprint']) && $_SESSION['fingerprint']!=$fingerprint)
    || isset($_GET['logout']) ) {
   do_logout();
}
session_regenerate_id();
$_SESSION['last_active'] = time();
$_SESSION['fingerprint'] = $fingerprint;
```

The do_logout() function destroys the session data and unsets the session cookie.

DATA BASES
MySQL

MySQL is the most popular database used with PHP. PHP with MySQL is a powerful combination showing the real power of Server-Side scripting. PHP has a wide range of MySQL functions available with the help of a separate module. In PHP5, this module has been removed and must be downloaded separately.

MySQL allows users to create tables, where data can be stored much more efficiently than the way data is stored in arrays.

In order to use MySQL or databases in general effectively, you need to understand SQL, or Structured Query Language.

Note that this page uses the mysqli functions and not the old mysql functions.

How to - Step By Step

Connecting to the MySQL server

PHP has the function mysqli_connect to connect to a MySQL server which handles all of the low level socket handling. We will supply 4 arguments; the first is the name of your MySQL server, the second a MySQL username, third a MySQL password and last a database name. In this example, it is assumed your server is localhost. If you are running a web server on one system, and MySQL on another system, you can replace localhost with the IP address or domain name of the system which MySQL resides on (ensure all firewalls are configured to open the appropriate ports). mysqli_connect returns a link_identifier that we can now use for communicating with the database. We will store this link in a variable called $link.

```
<?php
  $cxn = mysqli_connect ("localhost", "your_user_name", "your_password",
"database_name");
 ?>
```

Running a Query

We have connected to the mysql server and then selected the database we want to use, now we can run an SQL query over the database to select information, do an insert, update or delete. To do this we use mysqli_query. This takes two arguments: the first is our link_identifier and the second is an SQL query string. If we are doing a select sql statement mysqli_query generates a resource or the Boolean false to say our query failed, and if we are doing a delete, insert or update it generates a Boolean, true or false, to say if that was successful or not.

The basic code for running a query is the php function "mysqli_query($cxn, $query)". The "$query" argument is a MySQL query. The database argument is a database connection(here, the connection represented by $cxn). For example, to return the query "SELECT * FROM customers ORDER BY customer_id ASC", you could write

```
<?php
```

```
   mysqli_query($cxn, "SELECT * FROM customers ORDER BY customer_id ASC");
 ?>
```

However, this straightforward method will quickly become ungainly due to the length of MySQL queries and the common need to repeat the query when handling the return. All (or almost all) queries are therefore made in two steps. First, the query is assigned a variable (conventionally, this variable is named "$query" or "$sql_query" for purposes of uniformity and easy recognition), which allows the program to call simply "mysqli_query($cxn, $sql_query)".

```
$sql_query = "SELECT * FROM customers ORDER BY customer_id ASC";
```

Secondly, to handle the information returned from the query, practical considerations require that the information returned also be assigned to a variable. Again by convention rather than necessity (i.e. you could name it anything you wanted), this information is often assigned to "$result", and the function is called by the assignment of the variable.

It is important to understand that this code calls the function mysqli_query, in addition to assigning the return to a variable "$result". [NOTE: The queries that ask for information -- SELECT, SHOW, DESCRIBE, and EXPLAIN -- return what is called a resource. Other types of queries, which manipulate the database, return TRUE if the operation is successful and FALSE if not, or if the user does not have permission to access the table referenced.]

To catch an error, for debugging purposes, we can write:

```
<?php
   $result = mysqli_query ($cxn, $sql_query);
     or die (mysqli_error () . " The query was:" . $sql_query);
 ?>
```

If the function mysqli_query returns false, PHP will terminate the script and print an error report from MySQL (such as "you have an error in your SQL syntax") and the query.

Thus, our final code would be, assuming that there were a database connection named $cxn:

```
<?php
 $sql_query = "SELECT * FROM customers ORDER BY customer_id ASC";
 $result = mysqli_query ($cxn, $sql_query);
     or die (mysqli_error () . " The query was:" . $sql_query);
 ?>
```

Putting it all together

In the previous sections we looked at 3 commands, but not at how to use them in conjunction with each other. So let's take a look at selecting information for a table in our mysql database called MyTable, which is stored in a mysql database called MyDB.

```
<?php

        //Connect to the mysql server and get back our link_identifier
        $link = mysql_connect ("your_database_host", "your_user_name",
```

```
"your_password");

        //Now we select which database we would like to use
        mysql_select_db ("MyDB", $link);

        //Our SQL Query
        $sql_query = "Select * From MyTable";

        //Run our sql query
        $result = mysql_query ($sql_query, $link);

        //Close Database Connection
        mysql_close ($link);

?>
```

Getting Select Query Information

Well that doesn't help, because what are we to do with $result? Well when we do a select query we select out information from a database we get back what is known as a resource, and that is what is stored in $result, our resource identifier. A resource is a special type of PHP variable, but lets look at how to access information in this resource.

We can use a function called mysql_fetch_assoc it takes one parameter, our resource identifier $result, and generates an associative array corresponding to the fetched row. Each column in the table corresponds to an index with the same name. We can now extract out information and print it like so:

```
<?php
 //Connect to the mysql server and get back our link_identifier
 $link = mysql_connect("localhost", "your_user_name", "your_password")
        or die('Could not connect: ' . mysql_error());


 //Now we select which database we would like to use
 mysql_select_db("MyDB") or die('could not select database');

 //Our SQL Query
 $sql_query = "Select * From MyTable";

 //Run our sql query
 $result = mysql_query($sql_query)or die('query failed'. mysql_error());

 //iterate through result
 while($row = mysql_fetch_assoc($result))
 {
    //Prints out information of that row
    print_r($row);
    echo $row['foo'];
    //Prints only the column foo.
 }
```

```
// Free resultset (optional)
mysql_free_result($result);


//Close the MySQL Link
mysql_close($link);
?>
```

PHP + MySQL + Sphinx

Once you understand the basics of how MySQL functions with PHP you might want to start learning about full text search engines. Once your site gets large (millions of database records) MySQL queries will start to get painfully slow, especially if you use them to search for text with wildcards (ex:
WHERE content='%text%')

. There are many free/paid solutions to stop this problem.

A good open source full text search engine is Sphinx Search. There is a WikiBook on how to use it with PHP and MySQL that explains the concepts of how Indexing works. You might want to read it before reading the official documentation.

PostgreSQL

PostgreSQL is another popular database used with PHP.

The basic syntax of PostgreSQL is the same as that of MySQL, although some functions have been renamed:
mysqli_connect becomes pg_connect
mysql_select_db is deprecated; it is specified in the pg_connect syntax
mysqli_query becomes pg_query
mysql_error becomes pg_last_error
mysql_close becomes pg_close
mysql_fetch_assoc becomes pg_fetch_assoc
mysql_free_result becomes pg_free_result

PHP Data Objects

PHP Data Objects, also known as **PDO**, is an interface for accessing databases in PHP without tying code to a specific database. Rather than directly calling *mysql_*, *mysqli_*, and *pg_* functions, developers can use the PDO interface, simplifying the porting of applications to other databases.

How do I get it?

The PHP Data Objects extension is currently included by default with installations of PHP 5.1. It is available for users of PHP 5.0 through PECL, but does not ship with the base package.

PDO uses features of PHP that were originally introduced in PHP 5.0. It is therefore not available for users of PHP 4.x and below.

Differences between PDO and the mysql extension

PHP Data Objects has a number of significant differences to the MySQL interface used by most PHP applications on PHP 4.x and below:

- Object-orientation. While the *mysql* extension used a number of function calls that operated on a connection handle and result handles, the PDO extension has an object-oriented interface.
- Database independence. The PDO extension, unlike the *mysql* extension, is designed to be compatible with multiple databases with little effort on the part of the user, provided standard SQL is used in all queries.
- Connections to the database are made with a **Data Source Name**, or **DSN**. A DSN is a string that contains all of the information necessary to connect to a database, such as 'mysql:dbname=test_db'.

Intigration with (html,forms..etc)

Integrating PHP

"So," you say, "I now know the basics of this language. But, um... how do I use it?" I'm glad you asked. There are quite a few ways that PHP is used. You already know that you can call a script directly from a URL on your server. You can use PHP in more ways than that, though! Following are a few methods that PHP can be called.

Forms

Forms are, by far, the most common way of interacting with PHP. As we mentioned before, it is recommended that you have knowledge of HTML, and here is where we start using it. If you don't, just head to the HTML Wikibook for a refresher.

Form Setup

To create a form and point it to a PHP document, the HTML tag <form> is used and an action is specified as follows:

```
<form method="post" action="action.php">
    <!-- Your form here -->
</form>
```

Once the user clicks "Submit", the form body is sent to the PHP script for processing. All fields in the form are stored in the variables $_GET or $_POST, depending on the method used to submit the form.

The difference between the GET and POST methods is that GET submits all the values in the URL, while POST submits values transparently through HTTP headers. A general rule of thumb is if you are submitting sensitive data, use POST. POST forms usually provide more security.

Remember $_GET and $_POST are superglobal arrays, meaning you can reference them anywhere in a PHP script. For example, you don't need to call global $_POST or global $_GET to use them inside functions.

Example

Let's look at an example of how you might do this.

```
<!-- Inside enterlogin.html -->
<html>
<head>
 <title>Login</title>
</head>
<body>
 <form method="post" action="login.php">
  Please log in.<br/>
  Username: <input name="username" type="text" /><br />
  Password: <input name="password" type="password" /><br/>
  <input name="submit" type="submit" />
 </form>
</body>
```

</html>This form would look like the following: Please log in.
Username: …
Password: …
submit

And here's the script you'd use to process it (login.php):

```
<?php
  // Inside enterlogin.html
  if($_POST['username'] == "Spoom" && $_POST['password'] ==
"idontneednostinkingpassword")
  {
   echo("Welcome, Spoom.");
  }
  else
  {
   echo("You're not Spoom!");
  }
 ?>
```

Let's take a look at that script a little closer.

```
if($_POST['username'] == "Spoom" && $_POST['password'] ==
"idontneednostinkingpassword")
```

As you can see, $_POST is an array, with keys matching the names of each field in the form. For backward compatibility, you can also refer to them numerically, but you generally shouldn't as this method is much clearer. And that's basically it for how you use forms to submit data to PHP documents. It's that easy.

For More Information

PHP Manual: Dealing with Forms

PHP from the Command Line

Although PHP was originally created with the intent of being a web language, it can also be used for commandline scripting (although this is not common, because simpler tools such as the bash scripting are available).

Output

You can output to the console the same way you would output to a webpage, except that you have to remember that HTML isn't parsed (a surprisingly common error), and that you have to manually output newlines. The typical hello world program would look like this:

```
<?php
  print "Hello World!\n";
 ?>
```

Notice the newline character at the end-of-line - newlines are useful for making your output look neat and readable.

Input

PHP has a few special files for you to read and write to the command line. These files include the stdin, stdout and stderr files. To access these files, you would open these files as if they were actual files using fopen, with the exception that you open them with the special php:// "protocol", like this: $fp = fopen("php://stdin","r");

To read from the console, you can just read from stdin. No special redirection is needed to write to the console, but if you want to write to stderr, you can open it and write to it: $fp = fopen("php://stderr","w");

Bearing how to read input in mind, we can now construct a simple commandline script that asks the user for a username and a password to authenticate himself.

```
<?php
  $fp = fopen("php://stdin","r");

  print "Please authenticate yourself\n";
  print "Username: ";
  // rtrim to cut off the \n from the shell
  $user = rtrim(fgets($fp, 1024));
  print "Password: ";
  // rtrim to cut off the \n from the shell
  $pass = rtrim(fgets($fp, 1024));
  if (($user=="someuser") && ($pass=="somepass")) {
   print "Good user\n";
   // ... do stuff ...
  } else die("Bad user\n");
  fclose($fp);
 ?>
```

Note that this script just serves as an example as to how to utilise PHP for commandline programming - the code contained here demonstrates a very poor way to authenticate a user or to store a password. Remember that your PHP scripts are readable by others!