Introduction to PHP

A PHP file may contain text, HTML tags and scripts. Scripts in a PHP file are executed on the server.

What you should already know

Before you continue you should have some basic understanding of the following:

- ✍ WWW, HTML and the basics of building Web pages
- ✍ Some scripting knowledge

What is PHP?

- ? PHP stands for **H**ypertext **P**reprocessor
- ? PHP is a server-side scripting language, like ASP
- ? PHP scripts are executed on the server
- ? PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- ? PHP is an open source software (OSS)
- ? PHP is free to download and use

What is a PHP File?

- ? PHP files may contain text, HTML tags and scripts
- ? PHP files are returned to the browser as plain HTML
- ? PHP files have a file extension of ".php", ".php3", or ".phtml"

What is MySQL?

- ? MySQL is a small database server
- ? MySQL is ideal for small and medium applications
- ? MySQL supports standard SQL
- ? MySQL compiles on a number of platforms
- ? MySQL is free to download and use

PHP + MySQL

- ? PHP combined with MySQL are cross-platform (means that you can develop in Windows and serve on a Unix platform)

Why PHP?

- ? PHP runs on different platforms (Windows, Linux, Unix, etc.)
- ? PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- ? PHP is FREE to download from the official PHP resource: www.php.net
- ? PHP is easy to learn and runs efficiently on the server side

Where to Start?

- Install an Apache server on a Windows or Linux machine
- Install PHP on a Windows or Linux machine
- Install MySQL on a Windows or Linux machine

PHP Syntax

You cannot view the PHP source code by selecting "View source" in the browser - you will only see the output from the PHP file, which is plain HTML. This is because the scripts are executed on the server before the result is sent back to the browser.

Basic PHP Syntax

A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.

Below, we have an example of a simple PHP script which sends the text "Hello World" to the browser:

```
<html>
<body>
<?php echo "Hello World"; ?>
</body>
</html>
```

A PHP scripting block always starts with **<?php** and ends with **?>**. A PHP scripting block can be placed anywhere in the document.

Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

There are two basic statements to output text with PHP: **echo** and **print**. In the example above we have used the echo statement to output the text "Hello World".

Variables in PHP

All variables in PHP start with a $ sign symbol. Variables may contain strings, numbers, or arrays.

Below, the PHP script assigns the string "Hello World" to a variable called $txt:

```
<html>
<body>
<?php
$txt="Hello World";
echo $txt;
?>
</body>
</html>
```

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

```
<html>
<body>
<?php
$txt1="Hello World";
$txt2="1234";
echo $txt1 . " " . $txt2 ;
?>
</body>
</html>
```

The output of the script above will be: "Hello World 1234".

Comments in PHP

In PHP, we use // to make a single-line comment or /* and */ to make a large comment block.

```
<html>
<body>
<?php
//This is a comment
/*
This is
a comment
block
*/
?>
</body>
</html>
```

PHP Operators

Operators are used to operate on values.

PHP Operators

This section lists the different operators used in PHP.

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0

++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

PHP Conditional Statements

Conditional statements in PHP are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In PHP we have two conditional statements:

- **if (...else) statement** - use this statement if you want to execute a set of code when a condition is true (and another if the condition is not true)
- **switch statement** - use this statement if you want to select one of many sets of lines to execute

The If Statement

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

Syntax

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

Example

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

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

If more than one line should be executed when a condition is true, the lines should be enclosed within curly braces:

```
<html>
<body>
<?php
$x=10;
if ($x==10)
{
echo "Hello<br />";
echo "Good morning<br />";
}
?>
</body>
</html>
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

Syntax

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

Example

This is how it works: First we have a single expression (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The default statement is used if none of the cases are true.

```
<html>
<body>
<?php
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>
</body>
</html>
```

PHP Looping

Looping statements in PHP are used to execute the same block of code a specified number of times.

Looping

Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.

In PHP we have the following looping statements:

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

The while Statement

The while statement will execute a block of code **if and as long** a condition is true.

Syntax

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

Example

The following example demonstrates a loop that will continue to run as long as the variable i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>
<?php
$i=1;
while($i<=5)
{
echo "The number is " . $i . "<br />";
$i++;
}
?>
</body>
</html>
```

The do...while Statement

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

Syntax

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

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i while it has a value of less than 5:

```
<html>
<body>
<?php
$i=0;
do
{
$i++;
echo "The number is " . $i . "<br />";
}
while ($i<5);
?>
</body>
</html>
```

The for Statement

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

Syntax

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

Note: The for statement has three parameters. The first parameter is for initializing variables, the second parameter holds the condition, and the third parameter contains any increments required to implement the loop. If more than one variable is includedin either the initialization or the increment section, then they should be separated by commas. The condition must evaluate to true or false.

Example

The following example prints the text "Hello World!" five times:

```
<html>
<body>
<?php
for ($i=1; $i<=5; $i++)
{
echo "Hello World!<br />";
}
?>
</body>
</html>
```

The foreach Statement

Loops over the array given by the parameter. On each loop, the value of the current element is assigned to $value and the array pointer is advanced by one - so on the next loop, you'll be looking at the next element.

Syntax

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

Example

The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>
<?php
$arr=array("one", "two", "three");
foreach ($arr as $value)
{
echo "Value: " . $value . "<br />";
}
?>
</body>
</html>
```

8. Arrays

The most inconvenient thing about a variable is that you can only store one value at a time. Arrays are special types that allow variables to overcome this limitation, so you can store as many values as you want in the same variable. For example, instead of having two variables “$number1” and “$number2”, you could have an array “$numbers” that will hold both values. Imagine the same thing with ten numbers. What about a hundred? Because of the flexibility of the array, it can store two values or two hundred values, without having to define other variables. PHP indexes all the values within an array using a number or a string, so you will know which of the values you’re using.

Working with arrays is easy. You can process each item one after another, or you could just take one at random. Each item in an array is commonly referred to as an element. These elements can be accessed directly via their index, which can be either a number or a string. By default, PHP starts indexing elements numerically, from zero, and increments the element’s index with each new addition, so keep in mind that the index of the last elements in a numerically indexed array is always the total number of elements minus one. Indexing arrays by string can be useful in cases where you need to store both names and values.

There are two ways you can create an array: with the “array()” function or directly using the array identifier “[]”. You can use the “array()” function when you want to assign multiple elements to an array at a time. Don’t think that an array can only contain elements of a certain type (for example, only numbers). You can have numbers, strings and booleans in the same array, PHP won’t mind at all.

$names = array(“John”, 279, “Betty”, TRUE);

This creates an array called “$names” which holds the specified elements. You can access an array element by placing its index between square brackets, right after the array name. Not only you can retrieve a value this way, but you can also assign a value to that element.

print $names[2]; //outputs “Betty”

$names[3] = “Harrison”; //replaces TRUE with “Harrison”

Remember that PHP starts indexing from zero, therefore “$names[0]” is “John”, and so on - the index of any element always is one less than the element’s place in the list. Another way to define an array is using the array identifier “[]” in conjunction with the array name. You can also use this to add new elements in you have already created an array – either using the “array()” function or the array identifier.

$names[] = "John";

$names[] = "Mary";

$names[] = "Betty";

$names = array(“Harry”, “Samantha”, “Danny”);

There is no need to place any numbers between the square brackets, PHP takes care of the index number, so you don’t have to figure out which is the next available slot. This doesn’t mean that you cannot add numbers, but pay attention not to skip any of them, because PHP will initialize only the elements with the index number you specify.

Arrays indexed by strings, and not numbers, can prove to be useful when you need to access elements in array by name, and not by number. For example, if you have an address book, it would be much better to have a field called “name” or “address”, instead of a numeric field called “1” or “2”. You can define an associative array pretty much the same way you define a numerically indexed array.

$sysinfo = array(computer_name => “My computer”,

cpu_mhz => 2800,

memory_size => 512,

multimedia => TRUE);

Or you can use the array identifier:

$sysinfo[computer_name] = “My computer”;

$sysinfo[cpu_mhz] = 2800;

$sysinfo[memory_size] = 512;

$sysinfo[multimedia] = TRUE;

It’s good to know that an array element can itself be an array, so this enabled you to create sophisticated data structures called multidimensional arrays. Let’s say you have an address book, and you use an array to hold the information on the people you know (John, Mary, Betty, Harry). But on the other hand, each element must be a collection of a person’s details (first name, last name, telephone number). This is how you define it:

$address_book = array(

array(first_name => “John”, last_name => “Davis”, phone_number => “1234567”),

array(first_name => “Mary”, last_name => “Stewart”, phone_number => “1234568”),

array(first_name => “Betty”, last_name => “Willis”, phone_number => “1234569”),

array(first_name => “Harry”, last_name => “Miller”, phone_number => “1234560”));

You can access the data using two indices, the first one for “$address_book” – which is a numeric index, and the second one for the person’s details – which is a string index. The following text outputs the text “Mary”:

print $address_book[1][first_name];

There are a lot of functions to use with arrays. You can merge, slice, shift and sort arrays, by using the functions with the same name: “array_merge()”, “array_slice()”, “array_shift()”, and “sort()”. Sorting is definitely one of the most used functions. The “sort()” function can sort both alphabetically or numerically, depending on the array elements. Read the PHP documentation to learn how to use these functions.

PHP Functions

Functions are the most important part of any programming language, PHP included. You can say in a few words that functions are pieces of code that accept values and produce results. While there are functions that you don't need to supply any values to, a function which does nothing is pointless. Functions come in handy when you're writing repetitive code, and you're looking to use the same code in other scripts.

```
print "Welcome to my web-page!";
```

A function is a block of code that is not immediately executed, but can be called by your scripts when needed. Functions can either be built-in or defined by the user. PHP has a lot of built-in functions; one of them is "print()". In the previous examples we used it to output strings to the web-browser. The values used along with the function are called arguments. Arguments are included in the parentheses of a function call, but this doesn't always apply. For some build-in functions, for example "print()", you can choose to use them or not use them, PHP will accept it either way.

Different functions require different arguments; functions may ask for multiple arguments (which are separated by commas), but on the other hand, they could also be executed without any arguments, in which case you only include the parenthesis after the function name. It all depends on the function and what it does. For example, a function that calculates the values of a purchase based on the price and the number of purchased items will require two values. Let's assign the result to a variable:

```
$value = calculate_value($item_price, $number_of_items); //calls the "calculate_value()" function with two arguments
```

As you can see, functions require data to be passed to them, and then return a result based on that data. Most functions give you some information after their completion, even if that only tells you if their mission was successful (a boolean type result). This is not always the case. You don't have to use any arguments when you use a simple function that outputs the same message everytime it is called, for example "Thanks for visiting our web-site", and it also doesn't return any result. When declaring a function, note that you must not add a semicolon after the function statement:

```
function display_goodbye_message() //no semicolon here!

{

print "<h1>Thanks for visiting our web-site</h1>";

}
```

This will create a function that, whenever called, will output the same string. It requires no arguments, and it doesn't return anything. This isn't so useful, and you probably won't see many similar functions. Let's get back to our value calculating function. As you can see, you can use the "return" statement to assign a result to the function – in this case, the result will be the calculated value of the purchase (the price multiplied by the number of items purchased). This is how you define it:

```
function calculate_value($price, $items)

{

return ($price * $items); //returns a result

}
```

We can call functions dynamically the same way you can access dynamic variables. This means that you can treat an expression's name as a function name, and thus call it.

```
$function_name = "calculate_value";

$function_name(40, 5);
```

This is identical to:

```
calculate_value(40, 5);
```

When a function uses a variable, there are several issues you should be aware of. It's important to know that if you declare a variable inside a function, that variable will only be available to the rest of that function's code. This is called a local variable. If you try to use this variable in another function, it will not hold any value.

But sometimes you want to access some important information outside the function, without using an argument. One way to do this, you must the "global" statement.

```
$exchange_rate = 1.27;

function calculate_value_from_foreign($price, $items)

{

global $exchange_rate; //make "$exchange_rate" available to the function
```

```
return ($price * $items * $exchange_rate);

}
```

We expanded the previous script a little bit. You can see how we use “global” to inform the function that the “$exchange_rate” variable has been declared outside it. If we have hadn’t done this (try to comment the line), the function would have returned zero, because “$exchange_rate” would have been “NULL”. The function simply doesn’t know the value of “$exchange_rate” if you don’t tell it that it was assigned outside the function. Be careful, because if you modify the value of a variable within a function, that value will be changed for the following code after the call to the function; PHP uses the variable itself, and not a copy.

Another way to access the data outside the function, without using an argument is by using a reference to a variable as an argument. You can do this by placing an ampersand “&” in front of the argument, when you define the function. This way, the function will directly access the variable thru the reference, and the variable can be assigned and read at the same time.

```
$exchange_rate = 1.27;

function add_tax_and_calculate_value_from_foreign($price, $items, &$rate) //notice the ampersand “&”, telling PHP to treat the variable “$rate” as a reference

{

$rate .= 0.01; //adds a 0.01 tax

return ($price * $items * $exchange_rate);

}

add_tax_and_calculate_value_from_foreign(15, 4, $exchange_rate)

print $exchange_rate; //the new exchange rate is 1.28, because the function has incremented it
```

As you can see, PHP is very versatile. You can define functions within other functions, you can define a function based on a condition, you can use a variable number of arguments, you can use a default argument, and many more. Functions don’t act like variables; that is, you don’t have to first declare it then use it. You can declare a function and the very end of your script, and use it at the beginning, even though this is really not recommended. Before starting the actual execution of the script, PHP searches the script for functions, so it will know where to look for if you call one.

PHP Built-in Functions

The real power of PHP comes from its functions.

In PHP - there are more than 700 functions available.

PHP Functions

We will only list a few useful functions in this tutorial.

A complete list of PHP functions

PHP Information

The phpinfo() function is used to output PHP information.

This function is useful for trouble shooting, providing the version of PHP, and how it is configured.

The phpinfo() function options

Name	Description
INFO_GENERAL	The configuration line, php.ini location, build date, Web Server, System and more
INFO_CREDITS	PHP 4 credits
INFO_CONFIGURATION	Local and master values for php directives
INFO_MODULES	Loaded modules
INFO_ENVIRONMENT	Environment variable information
INFO_VARIABLES	All predefined variables from EGPCS (Environment, GET, POST, Cookie, Server)
INFO_LICENSE	PHP license information
INFO_ALL	Shows all of the above. This is the default value

Example

```
<html>
<body>
<?php
// Show all PHP information
phpinfo();
?>
<?php
```

```
// Show only the general information
phpinfo(INFO_GENERAL);
?>
</body>
</html>
```

PHP Server Variables

All servers hold information such as which URL the user came from, what's the user's browser, and other information. This information is stored in variables.

In PHP, the $_SERVER is a reserved variable that contains all server information. The $_SERVER is a global variable - which means that it's available in all scopes of a PHP script.

Example

The following example will output which URL the user came from, the user's browser, and the user's IP address:

```
<html>
<body>
<?php
echo "Referer: " . $_SERVER["HTTP_REFERER"] . "<br />";
echo "Browser: " . $_SERVER["HTTP_USER_AGENT"] . "<br />";
echo "User's IP address: " . $_SERVER["REMOTE_ADDR"];
?>
</body>
</html>
```

PHP Header() Function

The header() function is used to send raw HTTP headers over the HTTP protocol.

Note: This function must be called before anything is written to the page!

Example

The following example will redirect the browser to the following URL: http://www.w3schools.com/:

```
<?php
//Redirect browser
header("Location: http://www.w3schools.com/");
?>
<html>
<body>
......
</body>
</html>
```

Note: This function also takes a second parameter - an optional value of true or false to determine if the header should replace the previous header. Default is TRUE.

However, if you pass in FALSE as the second argument you can FORCE multiple headers of the same type.

Example

```
<?php
header("WWW-Authenticate: Negotiate");
header("WWW-Authenticate: NTLM", FALSE);
?>
<html>
<body>
......
</body>
</html>
```

PHP The Date() Function

The date() function is used to format a time or a date.

The Date() Function

The date() function is used to format a time or a date.

Syntax

```
string date (date_format[,int timestamp])
```

This function returns a string formatted according to the specified format.

Date Formats

The table below shows the characters that may be used in the format string:

Character	Description
a	"am" or "pm"
A	"AM" or "PM"
B	Swatch Internet time (000-999)
d	Day of the month with a leading zero (01-31)
D	Three characters that represents the day of the week (Mon-Sun)
F	The full name of the month (January-December)
g	The hour in 12-hour format without a leading zero (1-12)
G	The hour in 24-hour format without a leading zero (0-23)
h	The hour in 12-hour format with a leading zero (01-12)
H	The hour in 24-hour format with a leading zero (00-23)
i	The minutes with a leading zero (00-59)
I	"1" if the date is in daylights savings time, otherwise "0"
j	Day of the month without a leading zero (1-31)

l	The full name of the day (Monday-Sunday)
L	"1" if the year is a leap year, otherwise "0"
m	The month as a number, with a leading zero (01-12)
M	Three letters that represents the name of the month (Jan-Dec)
n	The month as a number without a leading zero (1-12)
O	The difference to Greenwich time (GMT) in hours
r	An RFC 822 formatted date (e.g. "Tue, 10 Apr 2005 18:34:07 +0300")
s	The seconds with a leading zero (00-59)
S	The English ordinal suffix for the day of the month (st, nd, rd or th)
t	The number of days in the given month (28-31)
T	The local time zone (e.g. "GMT")
U	The number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)
w	The day of the week as a number (0-6, 0=Sunday)
W	ISO-8601 week number of year, weeks starting on Monday
Y	The year as a 4-digit number (e.g. 2003)
y	The year as a 2-digit number (e.g. 03)
z	The day of the year as a number (0-366)

Examples

```
<?php
//Prints something like: Monday
echo date("l");
//Prints something like: Monday 15th of January 2003 05:51:38 AM
echo date("l dS of F Y h:i:s A");
//Prints something like: Monday the 15th
echo date("l \\t\h\e jS");
?>
```

PHP File Handling

PHP includes a lot of built-in functions for handling files and directories. You can read, write, delete, and get lots of information on files thru the use of these functions. Note that before you start working with files, you have to make sure that you have the right permissions that will allow you to manipulate them. So while you're trying to figure out why you can't delete a file, it may be because the server doesn't allow you to do that.

The creation and removal of files are completed with two functions: "touch()" and "unlink()". "touch()" searches if a file exists, and if it doesn't, it creates one, according to the specified filename; "unlink()" is used to remove the file passed on as an argument:

touch("newinfo.txt"); //create a new file, if it doesn't already exist

unlink(“oldinfo.txt”); //delete a file

Now that you know how to create a file, you should learn how to access it. This is done using the “fopen()” function, with requires a string containing the file path and a string containing the mode in which the file is about to be opened, which tells “fopen()” to open a file for reading, writing, or both. The complete list of the available modes can be found in the PHP documentation. “fopen()” return an integer, also known as a file pointer, which should be assigned to a variable. This will be used later on to work with the opened file. If a file cannot be open for whatever reason, “fopen()” returns FALSE. After you’re done with the file, you should remember to close it with “fclose()”, passing as an argument the file pointer.

$oldfp = fopen(“oldinfo.txt”, “r”); //opens a file for reading

if(!$fp = fopen(“newinfo.txt”, “w”)) //tries to open a file for writing

die(“Error on opening file!”); //terminates the execution of the script if it cannot open the file

fclose($oldfp);

The file may be opened in one of the following modes:

File Modes	Description
r	Read only. File pointer at the start of the file
r+	Read/Write. File pointer at the start of the file
w	Write only. Truncates the file (overwriting it). If the file doesn't exist, fopen() will try to create the file
w+	Read/Write. Truncates the file (overwriting it). If the file doesn't exist, fopen() will try to create the file
a	Append. File pointer at the end of the file. If the file doesn't exist, fopen() will try to create the file
a+	Read/Append. File pointer at the end of the file. If the file doesn't exist, fopen() will try to create the file
x	Create and open for write only. File pointer at the beginning of the file. If the file already exists, the fopen() call will fail and generate an error. If the file does not exist, try to create it
x+	Create and open for read/write. File pointer at the beginning of the file. If the file already exists, the fopen() call will fail and generate an error. If the file does not exist, try to create it

Reading from files is done with “fgets()”, which reads data from a file until it reaches a new line “\n”, an EOF (end-of-file), or a specified length, in this order. So if you specify 4096 bytes but “fgets()” first reaches a new line, it stops further reading. You should know that after each reading or writing to a file, PHP automatically

increments an index which holds the current position in the file. So if you have just opened a file, and then read 100 bytes, the next time you will want to read something else using the same file pointer, PHP will continue from where the 101st byte. The “fgetc()” is similar to “fgets()”, except that it returns only a single character from a file every time it is called. Because a character is always 1 byte in size, “fgetc()” doesn’t require a length argument.

```
$text = fgets($fp, 2000); //reads 2000 bytes at most

$chr = fgetc($fp); //reads 1 byte only
```

While you can read lines using “fgets()”, you need a way of telling when you have reached the end-of-file, so you won’t continue reading without any results. This is accomplished with the “feof()” functions, which returns “TRUE” or “FALSE”, weather the position of a file pointer is at the end of it. You now have enough information to read a file line by line:

```
$filename = "test.txt";

$fp = fopen($filename, "r") or die("Couldn’t open $filename");

while(!feof($fp))

{

$line = fgets($fp);

print "$line<br>";

}

fclose($fp);
```

While “fgets()” works well around text files, you may sometimes want more functionality for your script. The “fread()” functions returns the specified amount of data from a file, and doesn’t take into consideration the end-of-line “\n”, like “fgets()”. “fread()” also starts from the current file position, but if you’re not happy with it, you can always change it using the “fseek()” function.

```
fseek($fp, 100); //moves the file pointer to the 100th byte

print(fread($fp, 15)); //outputs 15 bytes
```

Any writing to a file is done with the use of “fwrite()” or “fputs()” functions, which both use a file pointer and a string to perform the writing:

fwrite($fp, “Hello from PHP!”);

fputs($fp, “Hello again from PHP!”);

So far, these functions were great with a single user. However, on public web-sites, with many users accessing your scripts at the same time, your files could quickly become corrupt. PHP offers the “flock()” function, which will lock a file and won’t allow other processes to write or read the file while the current process is running. “flock()” requires a file pointer and an integer to do this:

flock($fp1, 1); //shared lock – allows read, doesn’t allow write

flock($fp2, 2); //exclusive lock – doesn’t allow neither read, nor write

flock($fp3, 3); //release lock – releases a shared or exclusive lock

There are a lot of other functions that you can use with files like: testing functions – “file_exists()”, “is_file()”, “is_dir()”, “is_readable()”, “is_writeable()”, “is_executable()”; functions that return information on files: “filesize()”, “fileatime()”, “filemtime()”, “filectime()”. You can figure out what the function does by just reading its name; more information about them can be found in the PHP documentation.

Example

The example below reads a file character by character, until the end of file is true:

```
<?php
if (!($f=fopen("welcome.txt","r")))
exit("Unable to open file.");
while (!feof($f))
{
$x=fgetc($f);
echo $x;
}
fclose($f);
?>
```

PHP Forms

A very powerful feature of PHP is the way it handles HTML forms!

PHP Form Handling

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

Look at the following example of an HTML form:

```
<html>
<body>
<form action="welcome.php" method="POST">
Enter your name: <input type="text" name="name" />
Enter your age: <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

The example HTML page above contains two input fields and a submit button. When the user fills in this form and hits the submit button, the "welcome.php" file is called.

The "welcome.php" file looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old!
</body>
</html>
```

A sample output of the above script may be:

```
Welcome John.
You are 28 years old!
```

Here is how it works: The $_POST["name"] and $_POST["age"] variables are automatically set for you by PHP. The $_POST contains all POST data.

Note: If the method attribute of the form is GET, then the form information will be set in $_GET instead of $_POST.

PHP Cookies

A cookie is often used to identify a user.

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests for a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

How to Create a Cookie

The setcookie() function is used to create cookies.

Note: The setcookie() function must appear BEFORE the <html> tag.

Syntax

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

Example

The following example sets a cookie named "uname" - that expires after ten hours.

```
<?php
setcookie("uname", $name, time()+36000);
?>
<html>
<body>
<p>
A cookie was set on this page! The cookie will be active when
the client has sent the cookie back to the server.
</p>
</body>
</html>
```

How to Retrieve a Cookie Value

When a cookie is set, PHP uses the cookie name as a variable.

To access a cookie you just refer to the cookie name as a variable.

Tip: Use the isset() function to find out if a cookie has been set.

Example

The following example tests if the uname cookie has been set, and prints an appropriate message.

```
<html>
<body>
<?php
if (isset($_COOKIE["uname"]))
echo "Welcome " . $_COOKIE["uname"] . "!<br />";
else
echo "You are not logged in!<br />";
?>
```

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

PHP Include Files (SSI)

Server Side Includes (SSI) are used to create functions, headers, footers, or elements that will be reused on multiple pages.

Server Side Includes

You can insert the content of one file into another file before the server executes it, with the require() function. The require() function is used to create functions, headers, footers, or elements that will be reused on multiple pages.

This can save the developer a considerable amount of time. If all of the pages on your site have a similar header, you can include a single file containing the header into your pages. When the header needs updating, you only update the one page, which is included in all of the pages that use the header.

Example

The following example includes a header that should be used on all pages:

```
<html>
<body>
<?php require("header.htm"); ?>
<p>
Some text
</p>
<p>
Some text
</p>
</body>
</html>
```

Object Oriented Programming in PHP

One of the most common programming concepts in the world is OOP, which stands for Object-Oriented Programming. Using this technique, programmers manipulate objects, which are made of functions and variables, instead of manipulating the functions and variables themselves. Let's say that you develop an e-commerce web-site. You could use an object to manage a shopping cart, and assign the object different properties and methods, based on what you want it to do. Properties stand for variables holding the information about the object (for example: the name, items in the cart, total value of the items, etc.), and methods stand for the functions that can be used with the object (for example: add an item into the cart, remove an item, empty cart, etc.)

It sounds simple, doesn't it? Well, it really is. But for an object to be defined, you have to have a template on which you will define the object. This is where classes come in. A class is a blueprint for one or more objects. Therefore, an object is to a class what a variable is to a type. A class is a set of characteristics, and an object is entity that is defined based on those characteristics. Another example: let's think about an automobile class. Such a class could have a characteristic (property) called "color". All objects created based on this class would have such a characteristic, but some objects would initialize this property to "red", others to "blue", and so on. This means that the class only holds a definition, and the object holds the actual value.

You can declare a class by using the "class" keyword. Let's define a simple class:

```
class automobile_class

{

var $color; //the color of the car

var $max_speed; //the maximum speed

var $price; //the price of the car, in dollars


function is_cheap()

{

return ($this->price < 5000); //returns TRUE if the price is smaller than 5000 dollars

}

}
```

In this small example you can notice some of the most important aspects of a class. After the declaration of the class, you can see the variables used within the class, which are called properties. These are declared using the "var" statement. While they can be defined anywhere within the class, you should really define them at the very top, so you can better see the class' properties. The functions within the class are called methods; they're used to manipulate the class' properties and produce results. In that simple method you can see that when we use a class method or property, we must use the "->" operator. The keyword "this" tells PHP that the property of method belongs to the class being defined.

An object is a special variable that contains a bundle of other variables and functions; you always have to use a class upon which to create an object. But, unlike

a class, you won't need to write any code, nor you will see how the class actually works. While you may first think that this isn't so great, in fact this is one of the main concepts of object-oriented programming. You only have to create the class once, then you can create a zillion objects, in a zillion other projects.

While a class only exists in code and is considered to be a blueprint, an object exists in memory and is a working instance of a class. An instance of an object is created using the "new" statement along with the name of the class the object is based on. Let's return to our automobile class:

```
$car_object = new automobile_class();

$car_object->color = "red";

$car_object->price = 6000;

if($car_object->is_cheap())

{

print "This car is cheap!";

}

else

{

print "This car is expensive!";

}
```

You can see that we use the "->" operator to access and modify object's properties. After that, we use the same operator to call a method.

Perhaps the greatest benefit of object-oriented code is its reusability. Because the classes used to create objects are self-enclosed, they can be easily pulled from one project and inserted into another. Additionally, it is possible to create child classes that inherit and/or override the characteristics of their parents. This technique allows you to create more complex and specialized objects. Even if you start with a small class, you can develop it to a complex class by time, with adding more properties and objects to its children classes.

Expanding PHP Classes

While you can always modify a class to add more properties and methods, there is a better way to create a more complex class based on a simple one. Extending existing classes by creating new classes which inherit their parent class properties and methods may come in handy when you need to create more classes based on the initial one. This is also called a "parent-child" relationship, the reasons are obvious. Know that the parent class must be defined before the child class, so the order in which the classes are defined is important. Let's get back to our automobile class, and create a new class based on the original one:

```
class used_automobile_class extends automobile_class

{

var $owner;

var $is_price_negotiable;



function write_negotiable()

{

if ($this->is_price_negotiable)

{

print "The price is negotiable!";

}

else

{

print "The price is NOT negotiable!";

}

}

}
```

We have created a new class that inherits the all of the properties and methods of "automobile_class", and adds some new ones. While you can always add new properties and methods, it is not possible to remove any of the properties or methods defined in the parent class. You can override them, though. This means that a child class can redefine a method of its parent class, and any new objects created based on the child class will use the child's redefined method.

After reading this, you may wonder why you should expand existing classes, and not just rewrite the class and adapt it to the new requirements. The answer lies in flexibility. There are times when you start writing a class from scratch, add some more properties as needed and create this large class, but then you want to get back to one of the initial structures and redefine it some other way. This would not have happened if you had created children classes that expanded the original class.

If you have worked with OOP before on other programming languages, you may be wondering "how about the constructors and destructors?" If you don't know what I'm talking about, you should be interested to know that PHP allows you to automatically call a special method (also known as a constructor) when you create a new instance of the class using "new". Destructors, on the other hand, are special methods that are automatically called when an object is destroyed. Unfortunately, there are no destructors in PHP, but you may use "register_shutdown_function()" instead to simulate most effects of a destructor.

Creating a constructor is easy, you only have to create a method with the same name as the class. In the following example, we output a string and set a variable when an object is created based on this class:

```
class automobile_class

{

var $negotiable_price;

var $price; //the price of the car, in dollars



function automobile_class()

{

print "Object created!";

this->$negotiable_price = FALSE;

}

}
```

Please take notice of the fact that in PHP there are some class and methods names that are reserved to language. A small constraint is that you cannot name a class "stdClass", because this is internally used by PHP. Also, it is recommended that you don't use function names beginning with "__", they are "magical" to PHP.

Using Sessions

Besides cookies, there is another way to pass information to different web-pages: sessions. A session-enabled page allocates unique identifiers to users the first time they access the page, and then reassociates them with the previously allocated ones when they return to the page. Any global variables that were associated with the session will then become available to your code. The difference between sessions and cookies is that a session can hold multiple variables, and you don't have to set cookies for every variable. By default, the session data is stored in a cookie with an expiry date of zero, which means that the session only remains active as long as the browser. When you close the browser, all the stored information is lost. You can modify this behavior by changing the "session.cookie_lifetime" setting in "php.ini" from zero to whatever you want the cookie lifetime to be.

PHP uses several functions to deal with sessions. Before you start the actual work with sessions, you must explicitly start a session with "session_start()". If you want sessions to start automatically, you must enable the "session.auto_start" setting in PHP's configuration file. This way a session will be initiated for every PHP document. After you have started a session, you have access to the session ID via the "session_id()" function. After you're done with the session, you can destroy it using "session_destroy()". The following script always assigns a new session ID:

```
session_start(); //starts or resumes a function

print "Your session ID is: " . session_id(); //displays the session ID

session_destroy(); //ends the session; comment this line and the browser will output the same session ID as before
```

While accessing the unique identifier is a good start, the main objective of a session is to hold the values of variables. You must register variables to a session using the "session_register()" function before you try to read them on a session-enabled page. Also, you should remember that when you register a variable to a session, and then change the variable's value, the altered value will be reflected in the session file. Finally, remember that "session_register()" requires you to pass as an argument the variable name, not the variable itself:

```
session_start();
```

```
if(isset($stored_var))

{

print $stored_var; //this will not be displayed the first time you load the page, because you haven't registered the variable yet!

}

else

{

$stored_var = "Hello from a stored variable!";

session_register("stored_var"); //don't do this: session_register($session_var)

}
```

As you can seen, you can test if a variable is assigned using the "isset()" function. While this works with all of the script's variables, to only check variables registered with a session you should use "session_is_registered()" function. You must again pass as an argument a string containing the variable name, and the function will return TRUE if the variable has been registered within the session.

```
session_start();

if(session_is_registered("stored_var"))

{

print $stored_var;

}

else

{

$stored_var = "Hello from a stored variable!";

session_register("stored_var"); //don't do this: session_register($session_var)

}
```

As you've read before, "session_destroy()" is used to clean up the session variables, and end the session. However, "session_destroy()" does not destroy the session's variables, and they will remain accessible to the rest of the script in which "session_destroy()" is called.

```
session_start();

session_register("test");

$test = 12;

session_destroy();

print $test; // outputs 12
```

If you want to remove the registered variables, you need to use the session_unset() function. This destroys all variables associated with a session, both in the session file and within the script.

```
session_start();

session_register("test");

$test = 12;

session_unset(); //$test is destroyed

session_destroy();

print $test; //outputs nothing
```