PHP NOTES

PHP is a powerful tool for making dynamic and interactive Web pages.

PHP is the widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

In our PHP tutorial you will learn about PHP, and how to execute scripts on your server

Pre-requisites

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

- HTML/XHTML
- JavaScript

What is PHP?

- PHP stands for **P**HP: **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
- PHP is free to download and use

What is a PHP File?

- PHP files can 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 database server
- MySQL is ideal for both small and large 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 (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?

To get access to a web server with PHP support, you can:

- Install Apache (or IIS) on your own server, install PHP, and MySQL
- Or find a web hosting plan with PHP and MySQL support

PHP Syntax

PHP code is executed on the server, and the plain HTML result is sent to the browser.

Basic PHP Syntax

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

On servers with shorthand support enabled you can start a scripting block with <? and end with ?>.

For maximum compatibility, we recommend that you use the standard form (<?php) rather than the shorthand form.

```
<?php
?>
```

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>
```

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".

Note: The file must have a .php extension. If the file has a .html extension, the PHP code will not be executed.

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 Variables

A variable is used to store information.

Variables in PHP

Variables are used for storing a values, like text strings, numbers or arrays. When a variable is declared, it can be used over and over again in your script. All variables in PHP start with a $ sign symbol.

The correct way of declaring a variable in PHP:

```
$var_name = value;
```

New PHP programmers often forget the $ sign at the beginning of the variable. In that case it will not work.

Let's try creating a variable containing a string, and a variable containing a number:

```
<?php
$txt="Hello World!";
$x=16;
?>
```

PHP is a Loosely Typed Language

In PHP, a variable does not need to be declared before adding a value to it.

In the example above, you see that you do not have to tell PHP which data type the

variable is. PHP automatically converts the variable to the correct data type,

depending on its value.

In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it.

In PHP, the variable is declared automatically when you use it.

Naming Rules for Variables

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and
  _)
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore ($my_string), or with capitalization ($myString)

PHP String Variables

A string variable is used to store and manipulate text.

String Variables in PHP

String variables are used for values that contain characters.

In this chapter we are going to look at the most common functions and operators used to manipulate strings in PHP. After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.

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

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

The output of the code above will be:

```
Hello World
```

Now, lets try to use some different functions and operators to manipulate the string.

The Concatenation Operator

There is only one string operator in PHP.

The concatenation operator (.) is used to put two string values together.

To concatenate two string variables together, use the concatenation operator:

```
<?php
$txt1="Hello World!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

The output of the code above will be:

```
Hello World! What a nice day!
```

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

The strlen() function

The strlen() function is used to return the length of a

string. Let's find the length of a string:

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

The output of the code above will be:

```
12
```

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

The strpos() function

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

If a match is found, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

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

The output of the code above will be:

```
6
```

The position of the string "world" in our string is position 6. The reason that it is 6 (and not 7), is that the first position in the string is 0, and not 1.

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
%=	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 If...Else Statements

Conditional statements 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 the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if a condition is true and another code if the condition is false
- **if...elseif .. else statement** - use this statement to select one of several blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

The if Statement

Use the if statement to execute some code only if a specified condition is true.
Syntax:

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

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

```
<html>
<body>

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

</body>
</html>
```

Notice that there is no ..else.. in this syntax. You tell the browser to execute some code **only if the specified condition is true**.

The if...else Statement

Use the if . else statement to execute some code if a condition is true and another code if a condition is
false.

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 if a condition is true/false, the lines should be enclosed within curly braces:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>

</body>
</html>
```

The if...elseif.......else **Statement**

Use the if....elseifelse statement to select one of several blocks of code to be executed.

Syntax:

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

Example

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

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

PHP Switch Statement

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

Use the switch statement to select one of many blocks of code to be executed.

Syntax:

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

This is how it works: First we have a single expression *n* (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 no match is found.

```
<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 Arrays

An array stores multiple values in one single variable.

What is an Array?

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value. An array is a special variable, which can store multiple values in one single variable.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1="Saab";
$cars2="Volvo";
$cars3="BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array! An array can hold all your variable values under a single name. And you can access the values by referring to the array name. Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three kind of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

---

Numeric Arrays

A numeric array stores each array element with a

numeric index. There are two methods to create a

numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

2. In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

Example

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

```
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

The code above will output:

```
Saab and Volvo are Swedish cars.
```

Associative Arrays

An associative array, each ID key is associated with a value. When storing data about specific named values, a numerical array is not always the best way to do it. With associative arrays we can use the values as keys and assign values to them.

Example 1
In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Example 2
This example is the same as example 1, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

```
Peter is 32 years old.
```

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array
  (
  "Griffin"=>array
  (
  "Peter",
  "Lois",
  "Megan"
  ),
  "Quagmire"=>array
  (
  "Glenn"
  ),
  "Brown"=>array
  (
  "Cleveland",
  "Loretta",
  "Junior"
  )
  );
```

The array above would look like this if written to the output:

```
Array
(
[Griffin] => Array
  (
  [0] => Peter
  [1] => Lois
  [2] => Megan
  )
[Quagmire] => Array
  (
  [0] => Glenn
  )
[Brown] => Array
  (




  [0]  => Cleveland
  [1]  => Loretta
  [2]  => Junior
  )
)
```

Example 2

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] .
" a part of the Griffin family?";
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

PHP Looping - While Loops

Loops execute a block of code a specified number of times, or while a specified condition is true. Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a specified 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 Loop

The while loop executes a block of code while a condition is true.

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

Example

The example below defines a loop that starts with i=1. The loop will continue to run as long as 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>
```

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

The do...while Statement

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.

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

Example

The example below defines a loop that starts with i=1. It will then increment i with 1, and write some output. Then the condition is checked, and the loop will continue to run as long as i is less than, or equal to 5:

```
<html>
<body>

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

</body>
</html>
```

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

The for loop and the foreach loop will be explained in the next section

PHP Looping - For Loops

Loops execute a block of code a specified number of times, or while a specified condition is true.

The for Loop

The for loop is used when you know in advance how many times the script should run.

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

Parameters:

- *init*: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- *condition*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment*: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

Note: Each of the parameters above can be empty, or have multiple expressions (separated by commas).

Example

The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>

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

</body>
</html>
```

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

The foreach Loop

The foreach loop is used to loop through arrays.

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

For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

Example

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

```
<html>
<body>

<?php
$x=array("one","two","three");
foreach ($x as $value)
  {
  echo $value . "<br />";
  }
?>

</body>
</html>
```

Output:

```
one
two
three
```

PHP Functions

The real power of PHP comes from its functions.

In PHP, there are more than 700 built-in functions.

PHP Built-in Functions

PHP Functions

To keep the browser from executing a script when the page loads, you can put your script into a function. A function will be executed by a call to the function. You may call a function from anywhere within a page.

Create a PHP Function

A function will be executed by a call to the function.

```
function functionName()
{
code to be executed;
}
```

PHP function guidelines:

- Give the function a name that reflects what the function does
- The function name can start with a letter or underscore (not a number)

Example

A simple function that writes my name when it is called:

```
<html>
<body>

<?php
function writeName()
{
echo "Kai Jim Refsnes";
}

echo "My name is ";
writeName();
?>

</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes
```

PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable.

Parameters are specified after the function name, inside the parentheses.

Example 1

The following example will write different first names, but equal last name:

```
<html>
<body>

<?php
function writeName($fname)
{
echo $fname . " Refsnes.<br />";
}

echo "My name is ";
writeName("Kai Jim");
echo "My sister's name is ";
writeName("Hege");
echo "My brother's name is ";
writeName("Stale");
?>

</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes.
My brother's name is Stale Refsnes.
```

Example 2

The following function has two parameters:

```
<html>
<body>

<?php
function writeName($fname,$punctuation)
{
echo $fname . " Refsnes" . $punctuation . "<br />";
}

echo "My name is ";
writeName("Kai Jim",".");
echo "My sister's name is ";
writeName("Hege","!");
echo "My brother's name is ";
writeName("Ståle","?");
?>
</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes!
My brother's name is Ståle Refsnes?
```

PHP Functions - Return values

To let a function return a value, use the return statement.

```
<html>
<body>

<?php
function add($x,$y)
{
$total=$x+$y;
return $total;
}

echo "1 + 16 = " . add(1,16);
?>

</body>
</html>
```

Example

Output:

```
1 + 16 = 17
```

PHP Forms and User Input

The PHP $_GET and $_POST variables are used to retrieve information from forms, like user input.

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.

Example

The example below contains an HTML form with two input fields and a submit button:

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>

</body>
</html>
```

When a user fills out the form above and click on the submit button, the form data is sent to a PHP file, called "welcome.php":

"welcome.php" looks like this:

```
<html>
<body>

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

</body>
</html>
```

Output could be something like this:

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

Form Validation

User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.

You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

The built-in $_GET function is used to collect values in a form with method="get".

The $_GET Function

The built-in $_GET function is used to collect values from a form sent with method="get".

Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send (max. 100 characters).

```
<form action="welcome.php" method="get">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

Example

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

```
http://localhost/welcome.php?fname=Peter&age=37
```

The "welcome.php" file can now use the $_GET function to collect form data (the names of the form fields will automatically be the keys in the $_GET array):

```
Welcome <?php echo $_GET["fname"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```

When to use method="get"?

When using method="get" in HTML forms, all variable names and values are displayed in the URL.

Note1: This method should not be used when sending passwords or other sensitive information! However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

Note2: The get method is not suitable for large variable values; the value cannot exceed 100 characters. The built-in $_POST function is used to collect values in a form with method="post".

The $_POST Function

The built-in $_POST function is used to collect values from a form sent with method="post".

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

Note: However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the post_max_size in the php.ini file).

```
<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

Example

When the user clicks the "Submit" button, the URL will look like this:

```
http://localhost/welcome.php
```

The "welcome.php" file can now use the $_POST function to collect form data (the names of the form fields will automatically be the keys in the $_POST array):

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

When to use method="post"?

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send. However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

The PHP $_REQUEST Function

The PHP built-in $_REQUEST function contains the contents of both $_GET, $_POST, and $_COOKIE. The $_REQUEST function can be used to collect form data sent with both the GET and POST methods.

```
Welcome <?php echo $_REQUEST["fname"]; ?>!<br />
You are <?php echo $_REQUEST["age"]; ?> years old.
```

PHP File Handling

File handling is an important part of any web application. You often need to open and process a file for different tasks. PHP has several functions for creating, reading, uploading, and editing files. Be careful when manipulating files! When you are manipulating files you must be very careful. You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "**test.txt**", stored on the server, that looks like this:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success):

Example

```
<?php
echo readfile("test.txt");
?>
```

The readfile() function is useful if all you want to do is open up a file and read its contents.

PHP File Open/Read/Close - PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function. The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

```
<?php
$myfile = fopen("test.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("test.txt"));
fclose($myfile);
?>
```

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

Modes	Description
r	Open a file for read only. File pointer starts at the beginning of the file
w	Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	Creates a new file for write only. Returns FALSE and an error if file already exists
r+	Open a file for read/write. File pointer starts at the beginning of the file
w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

PHP Read File - fread()

The fread() function reads from an open file. The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "test.txt" file to the end:

```
fread($myfile, filesize("test.txt"));
```

PHP Close File - fclose()

The fclose() function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
$myfile = fopen("test.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

PHP Read Single Line - fgets()

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "test.txt" file:

Example

```
<?php
$myfile = fopen("test.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
```

```
?>
```

Note: After a call to the fgets() function, the file pointer has moved to the next line.

PHP Check End-Of-File - feof()

The feof() function checks if the "end-of-file" (EOF) has been reached. The feof() function is useful for looping through data of unknown length.

The example below reads the "test.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("test.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file. The example below reads the "test.txt" file character by character, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("test.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

Note: After a call to the fgetc() function, the file pointer moves to the next character.

PHP Create File - fopen()

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

Example

```
$myfile = fopen("testfile.txt", "w")
```

PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

PHP Write to File - fwrite()

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written. The example below writes a couple of names into a new file called "newfile.txt":

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Krishna Kumar\n";
fwrite($myfile, $txt);
$txt = "Sai Kumar\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "Krishna Kumar" and second contained "Sai Kumar".

After we finished writing, we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

```
Krishna Kumar
Sai Kumar
```

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Sai Charan\n";
```

```
fwrite($myfile, $txt);
$txt = "Raja Mouli\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote is present:

```
Sai Charan
Raja Mouli
```

PHP File Upload

With PHP, it is easy to upload files to the server. However, with ease comes danger, so always be careful when allowing file uploads!

Configure The "php.ini" File

First, ensure that PHP is configured to allow file uploads. In your "php.ini" file, search for the file_uploads directive, and set it to On:

file_uploads = On

Create The HTML Form

Next, create an HTML form that allow users to choose the image file they want to upload:

```
 <html>
<body>
<form action="upload.php" method="post" enctype="multipart/form-data">
  Select image to upload:
  <input type="file" name="fileToUpload" id="fileToUpload">
  <input type="submit" value="Upload Image" name="submit">
</form>
</body>
</html>
```

Some rules to follow for the HTML form above:

- Make sure that the form uses method="post"
- The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form
- Without the requirements above, the file upload will not work.
- Other things to notice:
- The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

The form above sends data to a file called "upload.php", which we will create next.

Create The Upload File PHP Script

The "upload.php" file contains the code for uploading a file:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
  $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
  if($check !== false) {
    echo "File is an image - " . $check["mime"] . ".";
    $uploadOk = 1;
  } else {
    echo "File is not an image.";
    $uploadOk = 0;
  }
}
?>
```

PHP script explained:

- $target_dir = "uploads/" - specifies the directory where the file is going to be placed
- $target_file specifies the path of the file to be uploaded
- $uploadOk=1 is not used yet (will be used later)
- $imageFileType holds the file extension of the file (in lower case)
- Next, check if the image file is an actual image or a fake image

Note: You will need to create a new directory called "uploads" in the directory where "upload.php" file resides. The uploaded files will be saved there.

Check if File Already Exists

Now we can add some restrictions. First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and $uploadOk is set to 0:

```
// Check if file already exists
if (file_exists($target_file)) {
  echo "Sorry, file already exists.";
  $uploadOk = 0;
}
```

Limit File Size

The file input field in our HTML form above is named "fileToUpload".

Now, we want to check the size of the file. If the file is larger than 600KB, an error message is displayed, and $uploadOk is set to 0:

```
// Check file size
if ($_FILES["fileToUpload"]["size"] > 600000) {
  echo "Sorry, your file is too large.";
  $uploadOk = 0;
}
```

Limit File Type

The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting $uploadOk to 0:

```
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
  echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
  $uploadOk = 0;
}
```

Complete Upload File PHP Script

The complete "upload.php" file now looks like this:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));

// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
  $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
  if($check !== false) {
    echo "File is an image - " . $check["mime"] . ".";
    $uploadOk = 1;
  } else {
    echo "File is not an image.";
    $uploadOk = 0;
  }
}

// Check if file already exists
if (file_exists($target_file)) {
  echo "Sorry, file already exists.";
  $uploadOk = 0;
}

// Check file size
```

```
if ($_FILES["fileToUpload"]["size"] > 500000) {
  echo "Sorry, your file is too large.";
  $uploadOk = 0;
}

// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
  echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
  $uploadOk = 0;
}

// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
  echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
  if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
    echo "The file ". htmlspecialchars( basename( $_FILES["fileToUpload"]["name"])). " has been
uploaded.";
  } else {
    echo "Sorry, there was an error uploading your file.";
  }
}
?>
```

PHP Cookies

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 a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the **setcookie()** function.

Syntax

setcookie(*name, value, expire, path, domain, secure, httponly*);

Only the *name* parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "Krishna Kumar". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the **isset()** function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Krishna Kumar";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>


<?php
if(!isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
  echo "Cookie '" . $cookie_name . "' is set!<br>";
  echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
```

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

Note: The value of the cookie is automatically URL encoded when sending the cookie, and automatically decoded when received (to prevent URL encoding, use setrawcookie() instead).

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the **setcookie()** function:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<?php
if(!isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
  echo "Cookie '" . $cookie_name . "' is set!<br>";
  echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
```

Delete a Cookie

To delete a cookie, use the **setcookie()** function with an expiration date in the past:

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
echo "Cookie 'user' is deleted.";
?>
```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
if(count($_COOKIE) > 0) {
  echo "Cookies are enabled.";
} else {
  echo "Cookies are disabled.";
}
?>
```

PHP Sessions

An amount of time the client is connected with the server is called as a session. Session data is stored in server. Session variables may be used to store some information at server end. This session data can be accessed among multiple pages in server i.e. session data can be shared between multiple web pages at server. Sessions are used to identify the clients by storing some data such as userid to identify different users at server side.

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

Tip: If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```
<?php
// Start the session
session_start();
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
```

Note: The session_start() function must be the very first thing in your document. Before any HTML tags.

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (**session_start()**).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

```
<?php
session_start();
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
```

Another way to show all the session variable values for a user session is to run the following code:

Example

```
<?php
session_start();
print_r($_SESSION);
```

```
?>
```

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example

```
<?php
session_start();
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use **session_unset()** and **session_destroy()**:

Example

```
<?php
session_start();
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>
```