Aplikasi Web
Pertemuan-6
Elemen – elemen dasar PHP

Introduction to PHP

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

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 (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

Installing Apache PHP and MySQL

PHP and MySQL are usually associated with LAMP (Linux, Apache, MySQL, PHP). However, most PHP developer (including me) are actually using Windows when developing the PHP application. So this page will only cover the WAMP (Windows, Apache, MySQL, PHP). You will learn how to install Apache, PHP, and MySQL under Windows platform.

The first step is to download the packages :

- Apache : www.apache.org
- PHP : www.php.net
- MySQL : www.mysql.com

You should get the latest version of each packages. As for the example in this tutorial i'm using Apache 2.0.50 (apache_2.0.50-win32-x86-no_ssl.msi), PHP 4.3.10 (php-4.3.10-Win32.zip) and MySQL 4.0.18 (mysql-4.0.18-win.zip).

Installing Apache

Installing apache is easy if you download the Microsoft Installer (.msi) package. Just double click on the icon to run the installation wizard. Click next until you see the Server Information window. You can enter localhost for both the Network Domain and Server Name. As for the administrator's email address you can enter anything you want.

I'm using Windows XP and installed Apache as Service so everytime I start Windows Apache is automatically started.

Click the
Next button and choose Typical installation. Click Next one more time and choose where you want to install Apache (I installed it in the default location C:\Program Files\Apache Group). Click the Next button and then the Install button to complete the installation process.
To see if you Apache installation was successful open up you browser and type http://localhost in the address bar. You should see something like this :

By default Apache's **document root** is set to **htdocs** directory. The document root is where you must put all your PHP or HTML files so it will be process by Apache (and can be seen through a web browser). Of course you can change it to point to any directory you want. The configuration file for Apache is stored in C:\Program Files\Apache Group\Apache2\conf\httpd.conf (assuming you installed Apache in

C:\Program Files\Apache Group) . It's just a plain text file so you can use Notepad to edit it.
For example, if you want to put all your PHP or HTML files in C:\www just find this line in the httpd.conf :

DocumentRoot "C:/Program Files/Apache Group/Apache2/htdocs"

and change it to :

DocumentRoot "C:/www"

After making changes to the configuration file you have to restart Apache (Start > Programs > Apache HTTP Server 2.0.50 > Control Apache Server > Restart) to see the effect.

Another configuration you may want to change is the **directory index.** This is the file that Apache will show when you request a directory. As an example if you type http://www.php-mysql-tutorial.com/ without specifying any file the index.php file will be automatically shown.

Suppose you want apache to use index.html, index.php or main.php as the directory index you can modify the DirectoryIndex value like this :

DirectoryIndex index.html index.php main.php

Now whenever you request a directory such as http://localhost/ Apache will try to find the index.html file or if it's not found Apache will use index.php. In case index.php is also not found then main.php will be used

Installing PHP

First, extract the PHP package (php-4.3.10-Win32.zip). I extracted the package in the directory where Apache was installed (C:\Program Files\Apache Group\Apache2). Change the new created directory name to php (just to make it shorter). Then copy the file php.ini-dist in PHP directory to you windows directory (C:\Windows or C:\Winnt depends on where you installed Windows) and rename the file to php.ini. This is the PHP configuration file and we'll take a look what's in it later on.

Next, move the php4ts.dll file from the newly created php directory into the sapi subdirectory. Quoting from php installation file you can also place php4ts.dll in other places such as :

- In the directory where apache.exe is start from (C:\Program Files\Apache Group\Apache2 \bin)
- In your %SYSTEMROOT%\System32, %SYSTEMROOT%\system and % SYSTEMROOT% directory.
  Note: %SYSTEMROOT%\System32 only applies to Windows NT/2000/XP)
- In your whole %PATH%

Modifying Apache Configuration
Apache doesn't know that you just install PHP. We need to tell Apache about PHP and where to find it. Open the Apache configuration file in C:\Program Files\Apache Group\Apache2\conf\httpd.conf and add the following three lines :

```
LoadModule php4_module php/sapi/php4apache2.dll
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
```

The first line tells Apache where to load the dll required to execute PHP and the second line means that every file that ends with .php should be processed as a PHP file. You can actually change it to anything you want like .html or even .asp! The third line is added so that you can view your php file source code in the browser window. You will see what this mean when you browse this tutorial and click the link to the example's source code like this one.

Now restart Apache for the changes to take effect (Start > Programs > Apache HTTP Server 2.0.50 > Control Apache Server > Restart) . To check if everything is okay create a new file, name it as test.php and put it in document root directory (C:\Program Files\Apache Group\Apache2\htdocs). The content of this file is shown below.

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

phpinfo() is the infamous PHP function which will spit out all kinds of stuff about PHP and your server configuration. Type http://localhost/test.php on your browser's address bar and if everything works well you should see something like this :

[Gambar Localhost]

Installing MySQL

First extract the package (mysql-4.0.18-win.zip) to a temporary directory, then run setup.exe. Keep clicking the next button to complete the installation. By default MySQL will be installed in C:\mysql.

Open a DOS window and go to C:\mysql\bin and then run mysqld-nt --console , you should see some messages like these :

```
C:\mysql\bin>mysqld-nt --console
InnoDB: The first specified data file .\ibdata1 did not exist:
InnoDB: a new database to be created!
040807 10:54:09 InnoDB: Setting file .\ibdata1 size to 10 MB
InnoDB: Database physically writes the file full: wait...
040807 10:54:11 InnoDB: Log file .\ib_logfile0 did not exist: new to be
created InnoDB: Setting log file .\ib_logfile0 size to 5 MB
InnoDB: Database physically writes the file full: wait...
040807 10:54:12 InnoDB: Log file .\ib_logfile1 did not exist: new to be
createdInnoDB: Setting log file .\ib_logfile1 size to 5 MB
InnoDB: Database physically writes the file full: wait...
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
040807 10:54:31 InnoDB: Started
mysqld-nt: ready for connections.
Version: '4.0.18-nt' socket: '' port: 3306
```

Now open another DOS window and type C:\mysql\bin\mysql
if your installation is successful you will see the MySQL client running :

```
C:\mysql\bin>mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.0.18-nt
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Type exit on the mysql> prompt to quit the MySQL client.
Now let's **install MySQL as a Service.** The process is simple just type mysqld-nt --install to install the service and net start mysql to run the service. But make sure to shutdown the server first using mysqladmin -u root shutdown

```
C:\mysql\bin>mysqladmin -u root shutdown
C:\mysql\bin>mysqld-nt --install
Service successfully installed.

C:\mysql\bin>net start mysql

The MySQL service was started successfully.


C:\mysql\bin>mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.0.18-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Modifying PHP Configuration File (php.ini)
PHP stores all kinds of configuration in a file called php.ini.You can find this file in the directory where you installed PHP. Sometimes you will need to modify this file for example to use a PHP extension. I won't explain each and every configuration available just the ones that often need modification or special attention.

Some of the configurations are :

1. register_globals
2. error_reporting and display_errors
3. extension and extension_path
4. session.save_path
5. max_execution_time

register_globals

Before PHP 4.2.0 the default value for this configuration is **On** and after 4.2.0 the default value is **Off.** The reason for this change is because it is so easy to write **insecure** code with this value on. So make sure that this value is Off in php.ini.

error_reporting and display_errors

Set the value to error_reporting = E_ALL during development but after production set the value to error_reporting = E_NONE .

The reason to use E_ALL during development is so you can catch most of the nasty bugs in your code. PHP will complain just about any errors you make and spit out all kinds of warning (for example if you're trying to use an uninitialized variable).

However, after production you should change the value to E_NONE so PHP will keep quiet even if there's an error in your code. This way the user won't have to see all kinds of PHP error message when running the script.

One important thing to note is that you will also need to set the value of display_erros to On. Even if you set error_reporting = E_ALL you will not get any error message (no matter how buggy our script is) unless display_errors is set to On.

extension and extension_path

PHP4 comes with about 51 extensions such as GD library (for graphics creation and manipulation), CURL, PostgreSQL support etc. These extensions are not turned on automatically. If you need to use the extension, first you need to specify the location of the extensions and then uncomment the extension you want.

The value of extension_path must be set to the directory where the extension is installed which is PHP_INSTALL_DIR/extensions, with PHP_INSTALL_DIR is the directory where you install PHP. For example I installed PHP in C:\Program Files\Apache Group\Apache2\php so the extensions path is :

extension_path = C:/Program Files/Apache Group/Apache2/php/extensions/

Don't forget to add that last slash or it won't work

After specifying the extension_path you will need to uncomment the extension you want to use. In php.ini a comment is started using a semicolon (;). As an example if you want to use GD library then you must remove the semicolon at the beginning of ; extension=php_gd2.dll to extension=php_gd2.dll

session.save_path

This configuration tells PHP where to save the session data. You will need to set this value to an existing directory or you will not be able to use session. In Windows you can set this value as session.save_path = c:/windows/temp/

max_execution_time

The default value for max_execution_time is 30 (seconds). But for some scripts 30 seconds is just not enough to complete it's task. For example a database backup script may need more time to save a huge database.

If you think your script will need extra time to finish the job you can set this to a higher value. For example to set the maximun script execution time to 15 minutes (900

seconds) you can modify the configuration as max_execution_time = 900

PHP have a convenient function to modify PHP configuration in runtime, ini_set(). Setting PHP configuration using this function **will not** make the effect permanent. It last only until the script ends.

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".

Opening & Ending PHP Tags

To open a block of PHP code in a page you can use one of these four sets of opening and closing tags

Opening Tag	Closing Tag
<?	?>
<?php	?>
<%	%>
<script language="php">	</script>

The first pair (**<?** and **?>**) is called short tags. You should avoid using short tags in your application especially if it's meant to be distributed on other servers. This is because short tags are not always supported (though I never seen any web host that don't support it). Short tags are only available only explicitly enabled setting the short_open_tag value to On in the PHP configuration file php.ini.

So, for all PHP code in this website I will use the second pair, **<?php** and **?>.**

Now, for the first example create a file named hello.php (you can use NotePad or your favorite text editor) and put it in your web servers root directory. If you use Apache the root directory is APACHE_INSTALL_DIR\htdocs, with APACHE_INSTALL_DIR is the directory where you install Apache. So if you install Apache on C:\Program Files\Apache Group\Apache2\htdocs then you put hello.php in C:\Program Files\Apache Group\Apache2\htdocs

Example : hello.php

```
<html>
<head>
<title>My First PHP Page</title>
</head>
<body>
<?php
echo "<p>Hello World, How Are You Today?</p>";
?>
</body>
</html>
```

To view the result start Apache then open your browser and go to http://localhost/hello.php

To view the result start Apache then open your browser and go to http://localhost/hello.php or http://127.0.0.1/hello.php. You should see something like this in your browser window.

The example above shows how to insert PHP code into an HTML file. It also shows the echo statement used to output a string. See that the echo statement ends with a semicolon. Every command in PHP must end with a semicolon. If you forget to use semicolon or use colon instead after a command you will get an error message like this

Parse error: parse error, unexpected ':', expecting ',' or ';' in c:\Apache\htdocs\examples\get.php on line 7

However in the hello.php example above omitting the semicolon won't cause an error. That's because the echo statement was immediately followed by a closing tag.

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".

Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is **case-sensitive**, so $myvar is different from $myVar.

A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

Example :

```
<?php
$myvar   = "Hello"; // valid
$yourVar_is-123 = "World"; // valid
$123ImHere = "Something"; // invalid, starts with number
?>
```

Variable Scope

The scope of a variable is the context within which it is defined. Basically you can not access a variable which is defined in different scope.

The script below will not produce any output because the function Test() declares no $a variable. The echo statement looks for a local version of the $a variable, and it has not been assigned a value within this scope. Depending on error_reporting value in php.ini the script below will print nothing or issue an error message.

Example :

```
<?php
$a = 1; // $a is a global variable

function Test()
{
echo $a; // try to print $a, but $a is not defined here
}

Test();
?>
```

If you want your global variable (variable defined outside functions) to be available in function scope you need to use the$global keyword. The code example below shows how to use the $global keyword.

Example :

```
<?php
$a = 1; // $a is defined in global scope ...
$b = 2; // $b too

function Sum()
{
  global $a, $b; // now $a and $b are available in Sum()
  $b = $a + $b;
}
Sum();
echo $b;
?>
```

PHP Superglobals

Superglobals are variables that available anywhere in the program code. They are :

- **$_SERVER**
  Variables set by the web server or otherwise directly related to the execution environment of the current script. One useful variable is $_SERVER ['REMOTE_ADDR'] which you can use to know you website visitor's IP address

  Your computer IP is
  ```
  <?php
  echo $_SERVER['REMOTE_ADDR'];
  ?>
  ```
- **$_GET**
  Variables provided to the script via HTTP GET. You can supply GET variables into a PHP script by appending the script's URL like this : http://www.php-mysql-tutorial.com/../examples/get.php?name=php&friend=mysql
  or set the a form method as method="get"

  example :
  ```
  <?php
  echo "My name is {$_GET['name']} <br>";
  echo "My friend's name is {$_GET['friend']}";
  ?>
  ```
  Note that I put $_GET['name'] and $_GET['friend'] in curly brackets. It's necessary to use these curly brackets when you're trying to place the value of an array into a string.

You can also split the string like this :
echo "My name is " . {$_GET['name']} . "
"; but it is easier to put the curly brackets.

- **$_POST**
  Variables provided to the script via HTTP POST. These comes from a form which set method="post"
- **$_COOKIE**
  Variables provided to the script via HTTP cookies.
- **$_FILES**
  Variables provided to the script via HTTP post file uploads. You can see the example in
  Uploading files to MySql Database
- **$_ENV**
  Variables provided to the script via the environment.
- **$_REQUEST**
  Variables provided to the script via the GET, POST, and COOKIE input mechanisms, and which therefore cannot be trusted. It's use the appropriate $_POST or $_GET from your script instead of using $_REQUEST so you will always know that a variable comes from POST or GET.
- **$GLOBALS**
  Contains a reference to every variable which is currently available within the global scope of the script.

You usually won't need the last three superglobals in your script.

Comments in PHP

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

Comment is a part of your PHP code that will not be translated by the PHP engine. You can use it to write documentation of your PHP script describing what the code should do. A comment can span only for one line or span multiple line.

PHP support three kinds of comment tags :

1. //
   This is a one line comment
2. #
   This is a Unix shell-style comment. It's also a one line comment
3. /* ..... */
   Use this multi line comment if you need to.

example :

```
<html>
<body>

<?php

//This is a comment

/*
This is
a comment
block
*/

?>

</body>
</html>
```

example :

```
<?php
echo "First line <br>"; // You won't see me in the output
// I'm a one liner comment
/*
Same thing, you won't
see this in the output file
*/
echo "The above comment spans two lines <br>";

# Hi i'm a Unix shell-style comment
# Too bad you can't see me
echo "View the source of this file, you'll see no comments here <br>";
?>
```

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

Variable Types

PHP supports eight primitive types.
Four scalar types:

- boolean : expresses truth value, TRUE or FALSE. Any non zero values and non empty string are also counted as TRUE.
- integer : round numbers (-5, 0, 123, 555, ...)
- float : floating-point number or 'double' (0.9283838, 23.0, ...)
- string : "Hello World", 'PHP and MySQL, etc

Two compound types:

- array
- object

And finally two special types:

- resource (one example is the return value of mysql_connect() function)
- NULL

In PHP an array can have numeric key, associative key or both. The value of an array can be of any type. To create an array use the array() language construct like this.

example : array.php

```
<?php
$numbers = array(1, 2, 3, 4, 5, 6);
$age = array("mom" => 45, "pop" => 50, "bro" => 25);
$mixed = array("hello" => "World", 2 => "It's two";

echo "numbers[4] = {$numbers[4]} <br>";
echo "My mom's age is {$age['mom']} <br>";
echo "mixed['hello'] = {$mixed['hello']} <br>";
echo "mixed[2] = {$mixed[2'}";
?>
```

When working with arrays there is one function I often used. The print_r() function. Given an array this function will print the values in a format that shows keys and elements

printr.php

```
<?php
$myarray = array(1, 2, 3, 4, 5);
$myarray[5] = array("Hi", "Hello", "Konnichiwa", "Apa Kabar");
echo '<pre>';
print_r($myarray);
echo '</pre>';
?>
```

Don't forget to print the preformatting tag <pre> and </pre> before and after calling print_r(). If you don't use them then you'll have to view the page source to see a result in correct format.

Type Juggling

In PHP you don't need to explicitly specify a type for variables. A variable's type is determined by the context in which that variable is used. That is to say, if you assign a string value to variable $var, $var becomes a string. If you then assign an integer value to $var, it becomes an integer.

An example of PHP's automatic type conversion is the addition operator '+'. If any of the operands is a float, then all operands are evaluated as floats, and the result will be a float. Otherwise, the operands will be interpreted as integers, and the result will also be an integer. Note that this does NOT change the types of the operands themselves; the only change is in how the operands are evaluated.

Example :

```
<?php
$myvar = "0"; // $myvar is string (ASCII 48)
$myvar += 2; // $myvar is now an integer (2)
$myvar = $foo + 1.3; // $myvar is now a float (3.3)
$myvar = 5 + "10 Piglets"; // $foo is integer (15)
?>
```

Type Casting

To cast a variable write the name of the desired type in parentheses before the variable which is to be cast.

Example : casting.php

```
<?php
$abc = 10; // $abc is an integer
$xyz = (boolean) $abc; // $xyz is a boolean

echo "abc is $abc and xyz is $xyz <br>";
?>
```

The casts allowed are:

- (int), (integer) - cast to integer
- (bool), (boolean) - cast to boolean
- (float), (double), (real) - cast to float
- (string) - cast to string
- (array) - cast to array
- (object) - cast to object

Playing With Strings

Strings are probably what you will use most in your PHP script. From concatenating, looking for patterns, trim, chop etc. So I guess it's a good idea to take a better look at this creature. We will also take a peek at some string functions that you might find useful for everyday coding.

Creating a string

To declare a string in PHP you can use double quotes (") or single quotes ('). There are some differences you need to know about using these two.

If you're using double-quoted strings variables will be expanded (processed). Special characters such as line feed (\n) and carriage return (\r) are expanded too. However, with single-quoted strings none of those thing happen. Take a look at the example below to see what I mean.

Note that browsers don't print newline characters (\r and \n) so when you open string.php take a look at the source and you will see the effect of these newline characters.

Example : string.php

```
<?php
$fruit = 'jamblang';

echo "My favourite fruit is $fruit <br>";
echo 'I lied, actually I hate $fruit <br>';

echo "\r\n My first line \r\n and my second line <br>\r\n";
echo ' Though I use \r\n this string is still on one line <br>';
?>
```

String Concatenation

To concat two strings you need the dot (.) operator so in case you have a long string and for the sake of readability you have to cut it into two you can do it just like the example below.

Actually if you need to write a loong string and you want to write it to multiple lines you don't need concat the strings. You can do it just like the second example below where $quote2 is split into three lines.

Example : concat.php

```
<?php
$quote1 = "Never insult Dumbledore " .
          "in front of me!";
$quote2 = "Nami,
you are
my nakama!";
echo $quote1 . "<br>";
echo $quote2;
?>
```

String Functions

substr($string, $start, $end) : get a chunk of $string

example : substring.php

```
<?php
// print '12'
echo substr('123456789', 0, 2);
// print '56789'
echo substr('123456789', 4);
// print '89'
echo substr('123456789', -2);
// print '456'
echo substr('123456789', 3, -4);
?>
```

str_repeat($string, $n) : repeat $string $n times

For example if you want to print a series of ten asteriks (*) you can do it with a for loop like this :

example : repeat01.php

```
<?php
for ($i = 0; $i < 10; $i++) {
   echo '*';
}
?>
```

Or you can go the easy way and do it like this :

example : repeat02.php

```
<?php
echo str_repeat('*', 10);
?>
```

strrchr($string, $char) : find the last occurence of the character $char in $string

For example: you want to get the file extension from a file name. You can use this function in conjunction with substr()

```
<?php
$ext = substr(strrchr($filename, '.'), 1);
?>
```

What the above code do is get a chunk of $filename starting from the **last dot** in $filename then get the substring of it starting from the second character (index 1).

To make things clearer suppose $filename is 'tutorial.php'. Using strrchr('tutorial.php', '.') yield '.php' and after substr('.php', 1) we get the file extension; 'php'

trim($string) : remove extra spaces at the beginning and end of $string

```
<?php
// print 'abc def'
echo trim('  abc def   ');
?>
```

explode($separator, $string) : Split $string by $separator

This function is commonly used to extract values in a string which are separated by a a certain separator string. For example, suppose we have some information stored as comma separated values. To extract each values we ca do it like shown below

```
<?php
// extract information from comma separated values
$csv = 'Uzumaki Naruto,15,Konoha Village';

$info = explode(',', $csv);
?>
```

Now, $info is an array with three values :

```
Array
(
  [0] => Uzumaki Naruto
  [1] => 15
  [2] => Konoha Village
)
```

We can further process this array like displaying them in a table, etc.

implode($string, $array) : Join the values of $array using $string

This one do the opposite than the previous function. For example to reverse back the $info array into a string we can do it like this :

```
<?php
$info = array('Uzumaki Naruto', 15, 'Konoha Village');

$csv = implode(',', $info);
?>
```

Another example : Pretend we have an array containing some values and we want to print them in an ordered list. We can use the implode() like this :

```
<?php
// print ordered list of names in array

$names = array('Uchiha Sasuke', 'Haruno Sakura', 'Uzumaki Naruto', 'Kakashi');

echo '<ol><li>' . implode('</li><li>', $names) . '</li></ol>';
?>
```

The result of that code is like an ordered list just like shown below

1. Uchiha Sasuke

2. Haruno Sakura
3. Uzumaki Naruto
4. Kakashi

By the way, i did write the above php code to print that list instead of writing the list directly