PHP Hypertext preprocessor

COURSE MATERIALS

You can access the course materials via this link

http://goo.gl/ev41na

CONTENTS-DAY01

- History of PHP.
- Why PHP?
- What do we need? (LAMP Overview)
- Installing LAMP
- PHP Overview (Variables, Constants, Flow control, ….)

HISTORY

1994

PHP originally stood for "personal home page". PHP development began by the Danish/Greenlandic programmer Rasmus Lerdorf

1997

Zeev Suraski and Andi Gutmans, two Israeli developers at the Technion IIT, rewrote the parser and formed the base of PHP 3, changed the name to PHP: Hypertext Preprocessor.

1999

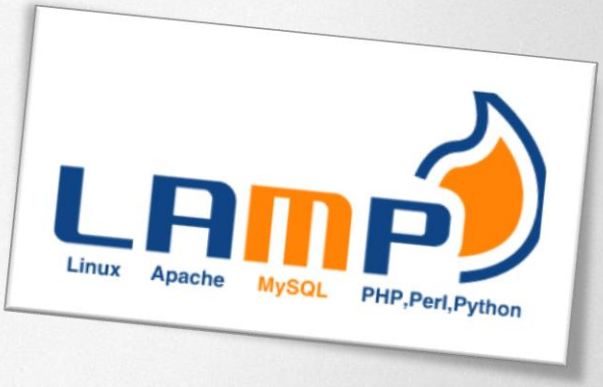
They started a new rewrite of PHP's core, producing the Zend Engine, They also founded Zend Technologies

WHY PHP?

- Ease of Learning PHP.
- Object-Oriented Support
- Portability
- Source Code
- Availability of Support and Documentation

WHAT DO WE NEED?

- **LAMP** is an acronym for a solution stack of free, open source software, originally coined from the first letters of Linux (operating system), Apache HTTP Server, MySQL (database software) and a Programming language like Perl/PHP/Python, principal components to build a viable general purpose web server.



INSTALLATION

- Ubuntu:

```
$ sudo tasksel install lamp-server
```

- CentOS:

```
$ sudo yum install httpd mariadb-server
mariadb php php-mysql
```

EMBEDDING PHP IN HTML

- Simply you can PHP in HTML page by Adding the php tag as the following:

```
<html>
<body>
<?php
echo '<h1>Hello, World!</h1>';
?>
</body>
</html>
```

- The PHP interpreter will run through the script and replace it with the output from the script.

PHP IS A SERVER SIDE

- The PHP has been interpreted and executed on the web server, as distinct from JavaScript and other client-side technologies interpreted and executed within a web browser on a user's machine.
- The code that you now have in this file consists of four types of text:
  - HTML
  - PHP tags
  - PHP statements
  - Whitespace

  You can also add comments.

PHP TAGS

- XML style

```
<?php echo '<p>Hello!.</p>'; ?>
```

- Short style

```
<? echo '<p>Hello!</p>'; ?>
```

- SCRIPT style

```
<script language='php'> echo
  '<p>Hello!.</p>'; </script>
```

- ASP style

```
<% echo '<p>Hello!.</p>'; %>
```

PHP TAGS

- Using XML style is recommended because it can't be closed off by the administrator beside it's portable through systems.
- Short Style is the simplest and follows the style of a Standard Generalized Markup Language (SGML) processing instruction. To use this type you need to enable the `short_open_tag` setting in your config file.
- Script Style This tag style is the longest and will be familiar if you've used JavaScript or VBScript.
- ASP Style is the same as used in Active Server Pages (ASP) or ASP.NET. You can use it if you have enabled the `asp_tags` configuration setting in php.ini.

PHP STATEMENTS & WHITESPACES

```
echo '<p>Hello, World!.</p>';
```

- Consists of reserved word to display content in browser, each line ends with (;)
- Spacing characters such as newlines (carriage returns), spaces, and tabs are known as whitespace. As you probably already know, browsers ignore whitespace in HTML. So does the PHP engine.
  - `echo 'hello ';`
  - `echo 'world';`

  and
  - `echo 'hello ';echo 'world';`
- are equivalent, but the first version is easier to read.

COMMENTS

- C-style, multiline comment that might appear at the start of a PHP script:

```
/* Author: Islam Askar
Last modified: June 24
This is to test comments!
*/
```

- You can also use single-line comments, either in the C++ style:

```
echo '<p>Hello.</p>'; // Comment
```

or in the shell script style:

```
echo '<p>Hello.</p>'; # Comment
```

ADDING DYNAMIC CONTENT

- We will put a function to print the Date and time of the machine

```
<?php
echo "<p>Now, It's ";
echo date('H:i, jS F Y');
echo "</p>";
?>
```

ACCESSING FORM VARIABLES

- You may be able to access the contents of the field in the following ways:
  - `$field_name` // short style
  - `$_POST['field_name']` // medium style
  - `$HTTP_POST_VARS['field_name']` // long style
- Short style ($ field_name) is convenient but requires the `register_globals` configuration setting be turned on.
- Medium style involves retrieving form variables from one of the arrays `$_POST`, `$_GET`, or `$_REQUEST`.

ACCESSING FORM VARIABLES

- Creating short variables name is recommended

```
<?php
// create short variable names
$field = $_POST['field'];
$field = $_GET['field'];
$field = $_REQUEST['field'];
?>
```

VARIABLES AND LITERALS

- Value itself is a literal.
- There are two kinds of strings:
  - Double quotation
  - Single quotation.
- PHP tries to evaluate strings in double quotation marks, resulting in the behavior shown earlier. Single-quoted strings are treated as true literals.

VARIABLES AND LITERALS

- There is also a third way of specifying strings using the heredoc syntax.
- Heredoc syntax allows you to specify long strings tidily, by specifying an end marker that will be used to terminate the string.

```
echo <<<theEnd
line 1
line 2
line 3
theEnd
```

UNDERSTANDING IDENTIFIERS

- Identifiers are the names of variables . You need to be aware of the simple rules defining valid identifiers:
  - Identifiers can be of any length and can consist of letters, numbers, and under-scores.
  - Identifiers cannot begin with a digit.
  - In PHP, identifiers are case sensitive. `$field` is not the same as `$Field` Trying to use them interchangeably is a common programming error. Function names are an exception to this rule: Their names can be used in any case.
  - A variable can have the same name as a function. This usage is confusing, however, and should be avoided. Also, you cannot create a function with the same name as another function.

EXAMINING VARIABLE TYPES

- A variable's type refers to the kind of data stored in it.
- PHP supports the following basic data types:
  - Integer—Used for whole numbers
  - Float (also called double)—Used for real numbers
  - String—Used for strings of characters
  - Boolean—Used for true or false values
  - Array—Used to store multiple data items
  - Object—Used for storing instances of classes

EXAMINING VARIABLE TYPES

- PHP is called weakly typed, or dynamically typed language. The type of a variable is determined by the value assigned to it.
- For example, when you created `$var1` and `$var2`, their initial types were determined as follows:
  - `$var1= 0;`
  - `$var2 = 0.00;`
- Strangely enough, you could now add a line to your script as follows:
  - `$var2 = 'Hello';`

EXAMINING VARIABLE TYPES

- You can pretend that a variable or value is of a different type by using a type cast. You simply put the temporary type in parentheses in front of the variable you want to cast.
- For example, you could have declared the two variables from the preceding section using a cast:
  - `$var1 = 0;`
  - `$var2 = (float)$var1;`

EXAMINING VARIABLE TYPES

- PHP provides one other type of variable: the variable variable.
- Variable variables enable you to change the name of a variable dynamically.
- For example, you could set

```
$varname = 'var1';
```

- You can then use `$$varname` in place of `$var1`. For example, you can set the value of `$var1` as follows:

```
$$varname = 5;
```

- This is exactly equivalent to

```
$var1= 5;
```

DECLARING CONSTANTS

- You can define these constants using the define function:

```
define('CONST1', 100);
```

- One important difference between constants and variables is that when you refer to a constant, it does not have a dollar sign in front of it. If you want to use the value of a constant, use its name only.

```
echo CONST1;
```

VARIABLE SCOPE

- The term scope refers to the places within a script where a particular variable is visible.
- The **six** basic scope rules in PHP are as follows:
  - Built-in superglobal variables are visible everywhere within a script.
  - Constants, once declared, are always visible globally; that is, they can be used inside and outside functions.
  - Global variables declared in a script are visible throughout that script, but not inside functions.
  - Global Variables inside functions refer to the global variables of the same name.
  - Static variables created inside functions are invisible from outside the function but keep their value between one execution of the function and the next.
  - Variables created inside functions are local to the function and cease to exist when the function terminates.

VARIABLE SCOPE

- Superglobals or autoglobals and can be seen everywhere, both inside and outside functions.
- The complete list of superglobals is as follows:
  - `$GLOBALS`—An array of all global variables (Like the global keyword, this allows you to access global variables inside a function—for example, as `$GLOBALS['myvariable']`.)
  - `$_SERVER`—An array of server environment variables
  - `$_GET`—An array of variables passed to the script via the GET method.
  - `$_POST`—An array of variables passed to the script via the POST method.
  - `$_REQUEST`—An array of all user input including the contents of input including $_GET, $_POST & $_COOKIE (but not $_FILES since PHP 4.3.0).
  - `$_COOKIE`—An array of cookie variables
  - `$_FILES`—An array of variables related to file uploads
  - `$_ENV`—An array of environment variables
  - `$_SESSION`—An array of session variables

OPERATORS AND PRECEDENCE

- Arithmetic operators are straightforward; they are just the normal mathematical operators.

Operator	Name	Example
+	Addition	$a + $b
-	Subtraction	$a - $b
*	Multiplication	$a * $b
/	Division	$a / $b
%	Modulus	$a % $b

- With each of these operators, you can store the result of the operation, as in this example:

```
$result = $a + $b;
```

OPERATORS AND PRECEDENCE

- You can use the string concatenation operator to add two strings and to generate and store a result much as you would use the addition operator to add two numbers:

```
$a = "Hello, ";
$b = "World!";
$result = $a.$b;
```

- The $result variable now contains the string "Hello, World!"

OPERATORS AND PRECEDENCE

- Combined assignment operators exist for each of the arithmetic operators and for the string concatenation operator

Operator	Use	Equivalent To
+=	$a += $b	$a=$a + $b
-=	$a -= $b	$a=$a - $b
*=	$a * =$b	$a=$a * $b
/=	$a / =$b	$a=$a / $b
%=	$a % =$b	$a=$a % $b
.=	$a.=$b	$a=$a.$b

OPERATORS AND PRECEDENCE

- The pre- and post-increment (++) and decrement (--) operators are similar to the +=and -= operators, but with a couple of twists.

```
$a=4;
echo ++$a;      //echo 5 , value of $a = 5

$a=4;
echo $a++;     //echo 4 , value of $a = 5
```

OPERATORS AND PRECEDENCE

- The reference operator (&, an ampersand) can be used in conjunction with assignment.

```
$a = 5;
$b = $a;
```

- These code lines make a second copy of the value in $a and store it in $b. If you subsequently change the value of $a, $b will not change:

```
$a = 7; // $b will still be 5
```

- You can avoid making a copy by using the reference operator. For example,

```
$a = 5;
$b = &$a;
$a = 7; // $a and $b are now both 7
```

OPERATORS AND PRECEDENCE

- References can be a bit tricky. Remember that a reference is like an **alias** rather than like a **pointer**. Both $a and $b point to the same piece of memory. You can change this by unsetting one of them as follows:

```
unset($a);
```

- Unsetting does not change the value of $b (7) but does break the link between $a and the value 7 stored in memory.

OPERATORS AND PRECEDENCE

- The comparison operators compare two values. Expressions using these operators return either of true or false.

Operator	Name	Use
==	Equals	$a == $b
===	Identical	$a === $b
!=	Not equal	$a != $b
!==	Not identical	$a !== $b
<>	Not equal	$a <> $b
<	Less than	$a < $b
>	Greater than	$a > $b
<= , >=	Less/greater than or equal to	$a <= $b

OPERATORS AND PRECEDENCE

- The logical operators combine the results of logical conditions. $a, is between 0 and 100. using the AND operator, as follows:

```
$a >= 0 && $a <=100
```

Operator	Name	Use	Result
!	NOT	!$b	Returns true if $b is false and vice versa
&&	AND	$a && $b	Returns true if both $a and $b are true; other-wise false
||	OR	$a || $b	Returns true if either $a or $b or both are true; otherwise false
and	AND	$a and $b	Same as &&, but with lower precedence
or	OR	$a or $b	Same as ||, but with lower precedence
xor	XOR	$a x or $b	Returns true if either $a or $b is true, and false if they are both true or both false.

OPERATORS AND PRECEDENCE

- The comma operator (,) separates function arguments and other lists of items. It is normally used incidentally.
- Two special operators, new and ->, are used to instantiate a class and access class members, respectively.
- The ternary operator (?:) takes the following form:

```
condition ? value if true : value if false
```

- This operator is similar to the expression version of an if-else statement, A simple example is

```
($grade >= 50 ? 'Passed' : 'Failed')
```

OPERATORS AND PRECEDENCE

- The error suppression operator (@) can be used in front of any expression—that is, any-thing that generates or has a value. For example,

```
$a = @(57/0);
```

- Without the @ operator, this line generates a divide-by-zero warning. With the operator included, the error is suppressed.
- The execution operator is really a pair of operators—a pair of backticks (``) in fact. The backtick is not a single quotation mark; it is usually located on the same key as the ~ (tilde) symbol on your keyboard.

```
$out = `ls -la`;
echo '<pre>'.$out.'</pre>';
```

OPERATORS AND PRECEDENCE

- There are a number of array operators. The array element operators ([]) enables you to access array elements.

Operator	Name	Use	Result
+	Union	$a+$b	Returns an array containing everything in $a and $b
==	Equality	$a == $b	Returns true if $a and $b have the same key and pairs
===	Identity	$a === $b	Returns true if $a and $b have the key and value pairs the same order
!=	Inequality	$a and $b	Returns true if $a and $b are not equal
<>	Inequality	$a or $b	Returns true if $a and $b are not equal
!==	Non-identity	$a x or $b	Returns true if $a and $b are not identical

OPERATORS AND PRECEDENCE

- There is one type operator: instanceof. This operator is used in object-oriented programming.
- The instanceof operator allows you to check whether an object is an instance of a particular class, as in this example:

```
class sampleClass{};
$myObject = new sampleClass();
if ($myObject instanceof sampleClass)
echo "myObject is an instance of
   sampleClass";
```

OPERATORS AND PRECEDENCE

Associativity	Operators
Left	,
Left	Or
Left	Xor
Left	And
Right	Print
Left	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
Left	: ?
Left	||
Left	&&
Left	|
Left	^
Left	&

OPERATORS AND PRECEDENCE

Associativity	Operators
n/a	== != === !==
n/a	< <= > >=
Left	<< >>
Left	+ - .
Left	* / %
Right	! ~ ++ -- (int) (double) (string) (array) (object) @
Right	[]
n/a	New
n/a	()

VARIABLE FUNCTIONS

- To use `gettype()`, you pass it a variable. It determines the type and returns a string containing the type name: bool, int, double (for floats), string, array, object, resource, or NULL. It returns unknown type if it is not one of the standard types.

```
string gettype(mixed var);
```

- `settype()`, you pass it a variable for which you want to change the type and a string containing the new type for that variable from the previous list.

```
bool settype(mixed var, string type);
```

VARIABLE FUNCTIONS

- `is_array()`—Checks whether the variable is an array.
- `is_double()`, `is_float()`, `is_real()` (All the same function)—Checks whether the variable is a float.
- `is_long()`, `is_int()`, `is_integer()` (All the same function)—Checks whether the variable is an integer.
- `is_string()`—Checks whether the variable is a string.
- `is_bool()`—Checks whether the variable is a boolean.

VARIABLE FUNCTIONS

- `is_object()`—Checks whether the variable is an object.
- `is_resource()`—Checks whether the variable is a resource.
- `is_null()`—Checks whether the variable is null.
- `is_scalar()`—Checks whether the variable is a scalar, that is, an integer, boolean, string, or float.
- `is_numeric()`—Checks whether the variable is any kind of number or a numericstring.
- `is_callable()`—Checks whether the variable is the name of a valid function.

VARIABLE FUNCTIONS

- `isset()` function takes a variable name as an argument and returns true if it exists and false otherwise. You can also pass in a comma-separated list of variables, and isset() will return true if all the variables are set.

```
bool isset(mixed var);[;mixed var[,...]])
```

- You can wipe a variable out of existence by using its companion function, unset(), which has the following prototype:

```
void unset(mixed var);[;mixed var[,...]])
```

VARIABLE FUNCTIONS

- `empty()` function checks to see whether a variable exists and has a nonempty, nonzero value; it returns true or false accordingly. It has the following prototype:

```
bool empty(mixed var);
```

FLOW CONTROL

- If – else – elseif

```
if (condition) {
  statement;
} elseif (condition) {
  statement;
} elseif (condition) {
  statement;
} else
{if (condition) {
  Statement;}
}
```

FLOW CONTROL

- Switch

```
switch($var) {
   case “value" :
     Statement;
     break;
   case " value " :
     Statement;
     break;
   default :
     Statement;
     break;
}
```

FLOW CONTROL

- While Loops

```
while( condition ) expression;
```

The following while loop will display the numbers from 1 to 5:

```
$num = 1;
while ($num <= 5 ){
echo $num.”<br />”;
$num++;
}
```

FLOW CONTROL

- for and foreach Loops

```
for( expression1; condition; expression2)
expression3;
```

  - expression1 is executed once at the start. Here, you usually set the initial value of a counter.
  - The condition expression is tested before each iteration. If the expression returns false, iteration stops. Here, you usually test the counter against a limit.
  - expression2 is executed at the end of each iteration. Here, you usually adjust the value of the counter.
  - expression3 is executed once per iteration. This expression is usually a block of code and contains the bulk of the loop code.

FLOW CONTROL

- do...while Loops

```
do
expression;
   while( condition );
```

- Example:

```
$num = 100;
do{
echo $num."<br />";
}while ($num < 1 ) ;
```

FLOW CONTROL

- use the `break` statement in a loop, execution of the script will continue at the next line of the script after the loop.
- If you want to jump to the next loop iteration, you can instead use the `continue` statement.
- If you want to finish executing the entire PHP script, you can use `exit`. This approach is typically useful when you are performing error checking

FLOW CONTROL

- For all the control structures we have looked at, there is an alternative form of syntax. It consists of replacing the opening brace ({) with a colon (:) and the closing brace with a new keyword, which will be `endif`, `endswitch`, `endwhile`, `endfor`, or `endforeach`, depending on which control structure is being used. No alternative syntax is available for do...while loops.

CONTENTS- DAY02

- Dealing with Files
- Arrays

STORING AND RETRIEVING DATA

- You can store data in two basic ways: in flat files or in a database.
- A flat file can have many formats, but in general, when we refer to a flat file, we mean a simple text file.

PROCESSING FILES

- Writing data to a file requires three steps:
  1. Open the file. If the file doesn't already exist, you need to create it.
  2. Write the data to the file.
  3. Close the file.

- Similarly, reading data from a file takes three steps:
  1. Open the file. If you cannot open the file (for example, if it doesn't exist), you need to recognize this and exit gracefully.
  2. Read data from the file.
  3. Close the file.

OPENING A FILE

- To open a file in PHP, you use the fopen() function. When you open the file, you need to specify how you intend to use it. This is known as the *file mode*.

```
resource fopen ( string $filename , string
$mode [, bool $use_include_path = false [,
resource $context ]] )
```

CHOOSING FILE MODES

- You need to make three choices when opening a file:
  1. You might want to open a file for reading only, for writing only, or for both reading and writing.
  2. If writing to a file, you might want to overwrite any existing contents of a file or append new data to the end of the file. You also might like to terminate your program gracefully instead of overwriting a file if the file already exists.
  3. If you are trying to write to a file on a system that differentiates between binary and text files, you might need to specify this fact.
- The `fopen()` function supports combinations of these three options.

USING fopen() TO OPEN A FILE

```
$fp=fopen("$DOCUMENT_ROOT/dir/file.txt",
'w');
```

- When fopen() is called, it expects two, three, or four parameters. Usually, you use two, as shown in this code line.
- As with the short names given form variables, you need the following line at the start of your script

```
$DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
```

USING `fopen()` TO OPEN A FILE

Mode	Mode Name	Result
r	Read	Open the file for reading, beginning from the start of the file.
r+	Read	Open the file for reading and writing, beginning from the start of the file.
w	Write	Open the file for writing, beginning from the start of the file. If the file already exists, delete the existing contents. If it does not exist, try to create it.
w+	Write	Open the file for writing and reading, beginning from the start of the file. If the file already exists, delete the existing contents. If it does not exist, try to create it.

USING fopen() TO OPEN A FILE

Mode	Mode Name	Result
x	Cautious write	Open the file for writing, beginning from the start of the file. If the file already exists, it will not be opened, fopen() will return false, and PHP will generate a warning.
x+	Cautious write	Open the file for writing and reading, beginning from the start of the file. If the file already exists, it will not be opened, fopen() will return false, and PHP will generate a warning.
a	Append	Open the file for appending (writing) only, starting from the end of the existing contents, if any. If it does not exist, try to create it.

USING fopen() TO OPEN A FILE

Mode	Mode Name	Result
a+	Append	Open the file for appending (writing) and reading, starting from the end of the existing contents, if any. If it does not exist, try to create it.
b	Binary	Used in conjunction with one of the other modes. You might want to use this mode if your file system differentiates between binary and text files. Windows systems differentiate; Unix systems do not. The PHP developers recommend you always use this option for maxi-mum portability. It is the default mode.
t	Text	Used in conjunction with one of the other modes. This mode is an option only in Windows systems. It is not recommended except before you have ported your code to work with the b option.

USING `fopen()` TO OPEN A FILE

- In addition to opening local files for reading and writing, you can open files via FTP, HTTP, and other protocols using `fopen()`.
- You can disable this capability by turning off the `allow_url_fopen` directive in the php.ini file. If you have trouble opening remote files with `fopen()`, check your php.ini file.

USING fopen() TO OPEN A FILE

- If the call to `fopen()` fails, the function will return false. You can deal with the error in a more user-friendly way by suppressing PHP's error message and giving your own:

```
@ $fp = fopen("$DOCUMENT_ROOT/dir/file.txt",
  'ab');
if (!$fp){
echo "<p><strong> Something went wrong!
  </strong></p></body></html>";
exit;
}
```

WRITING TO A FILE

- Writing to a file in PHP is relatively simple. You can use either of the functions `fwrite()` (file write) or `fputs()` (file put string); `fputs()` is an alias to `fwrite()`.
- The function `fwrite()` actually takes three parameters, but the third one is optional. The prototype for fwrite() is :

```
int fwrite ( resource handle, string string
  [, int length])
```

WRITING TO A FILE

- The third parameter, length, is the maximum number of bytes to write. If this parameter is supplied, `fwrite()` will write string until it reaches the end of string or has written length bytes, whichever comes first.
- You can obtain the string length by using `strlen()` function, as follows:

```
fwrite($fp, $outputstring,
  strlen($outputstring))
```

WRITING TO A FILE

- An alternative to `fwrite()` is the `file_put_contents()` function. It has the following prototype:

```
int file_put_contents ( string filename,
  string data[, int flags[, resource
  context]])
```

CLOSING A FILE

- After you've finished using a file, you need to close it. You should do this by using the `fclose()` function as follows:

```
fclose($fp);
```

- This function returns true if the file was successfully closed or false if it wasn't. This process is much less likely to go wrong than opening a file in the first place, so in this case we've chosen not to test it.

READING FROM A FILE

- You open the file by using fopen(). In this case, you open the file for reading only, so you use the file mode ‘rb’:

```
$fp = fopen(“$DOCUMENT_ROOT/dir/file.txt”,
  ‘rb’);
```

READING FROM A FILE

- `feof()` function takes a file handle as its single parameter. It returns true if the file pointer is at the end of the file. Although the name might seem strange, you can remember it easily if you know that *feof* stands for *File End Of File*.

```
while (!feof($fp)){
$text= fgets($fp, 999) }
```

- This function reads one line at a time from a file. In this case, it reads until it encounters a newline character (\n), encounters an EOF, or has read 998 bytes from the file. The maximum length read is the length specified minus 1 byte.

READING FROM A FILE

- An interesting variation on `fgets()` is `fgetss()`, which has the following prototype:

```
string fgetss(resource fp, int length,
  string [allowable_tags]);
```

- This function is similar to `fgets()` except that it strips out any PHP and HTML tags found in the string. If you want to leave in any particular tags, you can include them in the `allowable_tags` string

READING FROM A FILE

- `fgetcsv()` is another variation on `fgets()`. It has the following proto-type:

```
array fgetcsv ( resource fp, int length [,
  string delimiter[, string enclosure]])
```

- This function breaks up lines of files when you have used a delimiting character, such as the tab character or a comma. If you want to reconstruct the variables from the order separately rather than as a line of text, `fgetcsv()` allows you to do this simply. You call it in much the same way as you would call `fgets()`, but you pass it the delimiter you used to separate fields. For example,

```
$text = fgetcsv($fp, 100, "\t");
```

READING FROM A FILE

- Instead of reading from a file a line at a time, you can read the whole file in one go. There are four different ways:
- The first uses `readfile()`. You can replace almost the entire script you wrote previously with one line:

```
readfile("$DOCUMENT_ROOT/dir/file.txt");
```

READING FROM A FILE

- Second, use you can `fpassthru()`. You need to open the file using `fopen()` first. You can then pass the file pointer as an argument to `fpassthru()`, which dumps the contents of the file. It closes the file when it is finished.
- You can replace the previous script with `fpassthru()` as follows:

```
$fp = fopen
  (“$DOCUMENT_ROOT/dir/file.txt,‘rb’);

fpassthru($fp);
```

READING FROM A FILE

- The third option for reading the whole file is using the `file()` function. This function is identical to `readfile()` except that instead of echoing the file to standard out-put, it turns it into an array.

```
$filearray =
file($DOCUMENT_ROOT/dir/file.txt”);
```

READING FROM A FILE

- The fourth option is to use the `file_get_contents()` function. This function is identical to `readfile()` except that it returns the content of the file as a string instead of outputting it to the browser.

```
$file_content = file_get_contents
($filename);
```

OTHER USEFUL FILE FUNCTIONS

- Navigating Inside a File: `rewind()`, `fseek()`, and `ftell()`.
- The `rewind()` function resets the file pointer to the beginning of the file.
- The `ftell()` function reports how far into the file the pointer is in bytes. You can use the function `fseek()` to set the file pointer to some point within the file.

```
int fseek ( resource fp, int offset [, int
  whence])
```

OTHER USEFUL FILE FUNCTIONS

- Check type of file

```
String filetype(string $filepath)
```

- Change file mode

```
bool chmod ( string $filename , int $mode )
```

- Change file owner

```
bool chown ( string $filename , mixed $user )
```

- Change file group

```
bool chgrp ( string $filename , mixed $group
)
```

OTHER USEFUL FILE FUNCTIONS

- Checking Whether a File Is There: `file_exists()`.
- Determining How Big a File Is: `filesize()`.
- Deleting a File: `unlink()`.
- Copy file: `copy(string $source, string $destination)`

`is_dir()`

`is_executable()`

`is_file()`

`is_link()`

`is_readable()`

`is_writable()`

`basename()` function returns the filename from a path.

LOCKING FILES

- To lock a file while you are writing to it. Use `flock()` function. This function should be called after a file has been opened but before any data is read from or written to the file.
- The prototype for `flock()` is

```
bool flock (resource fp, int operation [, int
  &wouldblock])
```

LOCKING FILES

- You need to pass it a pointer to an open file and a constant representing the kind of lock you require. It returns true if the lock was successfully acquired and false if it was not.
- The optional third parameter will contain the value true if acquiring the lock would cause the current process to block (that is, have to wait).

Value of Operation	Meaning
LOCK_SH	Reading lock. The file can be shared with other readers.
LOCK_EX	Writing lock. This operation is exclusive; the file cannot be shared
LOCK_UN	The existing lock is released.

PROBLEMS WITH USING FLAT FILES

There are a number of problems in working with flat files:

- When a file grows large, working with it can be very slow.
- Searching for a particular record or group of records in a flat file is difficult. If the records are in order.
- Beyond the limits offered by file permissions, there is no easy way of enforcing different levels of access to data.
- Dealing with concurrent access can become problematic. You can lock files, but it can also cause a bottleneck.
- All of these file processing operations are sequential processing; you start from the beginning of the file and read through to the end.

INDEXED ARRAYS

- The indices of numerically indexed arrays in PHP, start at zero by default, although you can alter this value.

```
$array = array( ‘text1’, ‘text2’,‘text3’ );
```

- If you want an ascending sequence of numbers stored in an array, you can use the `range()` function to automatically create the array.

```
$numbers = range(1,10);
```

INDEXED ARRAYS

- `range()` function has an optional third parameter that allows you to set the step size between values.

```
$odds = range(1, 10, 2);
```

- The `range()` function can also be used with characters, as in this example:

```
$letters = range('a', 'z');
```

USING LOOPS

- Because the array is indexed by a sequence of numbers, you can use a for loop to more easily display its contents:

```
for ($i = 0; $i<count($array); $i++) {
echo $array[$i]." ";
}
```

USING LOOPS

- You can also use the `foreach` loop, specially designed for use with arrays. In this example, you could use it as follows:

```
foreach ($array as $current) {
echo $current." ";
}
```

ASSOCIATIVE ARRAYS

- Associative arrays are key indexed arrays, The following code creates an array with alphabets names as keys and orders as values:

```
$alphabets = array('a'=>1, 'b'=>2,
  'c'=>3);
```

- The following code creates the same `$alphabets`

```
$ alphabets = array( 'a'=>1 );
$ alphabets['b'] = 2;
$ alphabets['c'] = 3;
```

ASSOCIATIVE ARRAYS

- To create an array of variables:

```
$username = 'islam';

$email = 'isalah@iti.gov.eg';


$variables = compact('username',
'email');

// $variables is now ['username' =>
'islam', 'email' =>
'isalah@iti.gov.eg']
```

USING LOOPS

- Because the indices in an array are not numbers, you cannot use a simple counter in a for loop to work with the array. However, you can use the `foreach` loop or the `list`() and `each`() constructs

```
foreach ($alphabets as $key => $value)
  {
echo $key." - ".$value."<br />";
}
```

USING LOOPS

- The following code lists the contents of the `$alphabets` array using the `each()` construct:

```
while ($element = each($ alphabets )) {
echo $element['key'];
echo " – ";
echo $element['value'];
echo "<br />";
}
```

ARRAY OPERATORS

- One set of special operators applies only to arrays. Most of them have an analogue in the scalar operators.

Operator	Name	Use	Result
+	Union	$a+$b	Returns an array containing everything in $a and $b
==	Equality	$a == $b	Returns true if $a and $b have the same key and pairs
===	Identity	$a === $b	Returns true if $a and $b have the key and value pairs the same order
!=	Inequality	$a and $b	Returns true if $a and $b are not equal
<>	Inequality	$a or $b	Returns true if $a and $b are not equal
!==	Non-identity	$a x or $b	Returns true if $a and $b are not identical

MULTIDIMENSIONAL ARRAYS

- Arrays do not have to be a simple list of keys and values; each location in the array can hold another array. This way, you can create a two-dimensional array.

```
$alphabets=
array( array( '1st', 'a', 1 ),
       array( '2nd', 'b', 2 ),
          array( '3rd', 'c', 3 ) )
```

SORTING ARRAYS

- The following code showing the `sort()` function results in the array being sorted into ascending alphabetical order:

```
$names = array( 'Mostafa', 'Abdelrhuman',
  'Islam' );

sort($names);
```

- The array elements will now appear in the order Abdelrhuman, Islam, Mostafa.
- You can sort values by numerical order, too.

SORTING ARRAYS

- Note that the `sort()` function is case sensitive. All capital letters come before all lowercase letters. So A is less than Z, but Z is less than a.
- The function also has an optional second parameter. You may pass one of the constants `SORT_REGULAR` (the default), `SORT_NUMERIC`, or `SORT_STRING`.

USING asort() AND ksort()

```
$prices = array( 'meat'=>100, 'sugar'=>10,
  'tea'=>8 );

asort($prices);
```

- The function `asort()` orders the array according to the value of each element.
- If, instead of sorting by value, you want to sort by key, you can use `ksort()`

```
ksort($prices);
```

SORTING IN REVERSE

- The three different sorting functions—`sort()`, `asort()`, and `ksort()`—sort an array into ascending order.
- Each function has a matching reverse sort function to sort an array into descending order. The reverse versions are called `rsort()`, `arsort()`, and `krsort()`.

USER-DEFINED SORTING

- The user-defined sorts do not have reverse variants. Because you provide the comparison function, you can write a comparison function that returns the opposite values. To sort into reverse order, the function needs to return 1 if $x is less than $y and -1 if $x is greater than $y. For example,

```
function reverse_compare($x, $y) {
if ($x[1] == $y[1]) {
return 0;
```

USER-DEFINED SORTING

```
} else if ($x[1] < $y[1]) {
return 1;
} else {
return -1;
}
}
```

- Calling `usort($alphabets, 'reverse_compare')` would now result in the array being placed in descending order.
- The `uasort()` and `uksort()` versions of asort and ksort also require user-defined comparison functions.

REORDERING ARRAYS

- You might want to manipulate the order of the array in other ways.
- The function `shuffle()` randomly reorders the elements of your array.
- The function `array_reverse()` gives you a copy of your array with all the elements in reverse order.
- Use `array_push()` for each element to add one new element to the end of an array. As a side note, the opposite is `array_pop()`. This function removes and returns one element from the end of an array.

LOADING ARRAYS FROM FILES

- we can load data into array using the function `file()`. Each line in the file becomes one element of an array.
- We can also use the `count()` function to see how many elements are in an array.
- you can use the function `explode()` to split up each line into array

```
array explode(string separator, string string
[, int limit])
```

ARRAYS NAVIGATIONS

- We can combine elements of an Array using implode()

```
string implode ( string $glue, array $pieces
).
```

- To check if the value exists use: `in_array()`.

```
$fruits = ['banana', 'apple'];

$foo = in_array('banana', $fruits);
```

- Every array has an internal pointer that points to the current element in the array. If you create a new array, the current pointer is initialized to point to the first element in the array. Calling `current($array_name)` returns the first element.

ARRAYS NAVIGATIONS

- Calling either `next()` or `each()` advances the pointer forward one element. Calling `each( $array_name )` returns the current element before advancing the pointer.
- The function `next()` behaves slightly differently: Calling `next($array_name)` advances the pointer and then returns the new current element.

ARRAYS NAVIGATIONS

- `reset()` returns the pointer to the first element in the array. Similarly, calling `end( $array_name )` sends the pointer to the end of the array. The first and last elements in the array are returned by `reset()` and `end()`, respectively.
- To move through an array in reverse order, you could use `end()` and `prev()`.
- The `prev()` function is the opposite of `next()`. It moves the current pointer back one and then returns the new current element.

ARRAYS NAVIGATIONS

- For example, the following code displays an array in reverse order:

```
$value = end ($array);
while ($value){
echo "$value<br />";
$value = prev($array);
}
```

ARRAYS NAVIGATIONS

- Sometimes you might want to work with or modify every element in an array in the same way. The function `array_walk()` allows you to do this. The prototype is as follows:

```
bool array_walk(array arr, string func,
[mixed userdata])
```

Similar to the way you called `usort()` earlier, `array_walk()` expects you to declare a function of your own for example :

```
function my_print($value){
echo "$value<br />";
}
array_walk($array, 'my_print');
```

MANIPULATING AN ARRAY

- To merge to arrays use `array_merge();`
- We can convert an associative array into indexed using `array_merge():`

```
$a=array(3=>"red",4=>"green");

print_r(array_merge($a));
```

- `array_chunk()` splits an array into chunks

```
$input_array = array('a', 'b', 'c', 'd',
'e');

$output_array = array_chunk($input_array, 2);
```

MANIPULATING AN ARRAY

- To loop on two arrays at once (assuming both have the same length)

```
$people = [‘Islam', ‘Ahmed', ‘Sayed'];

$foods = ['chicken', 'beef', ‘meat'];

array_map(function($person, $food) {

    return "$person likes $food\n";

}, $people, $foods);
```

- We can merge to indexed arrays into one associative array using `array_combine()`:

```
$combinedArray = array_combine($people,
$foods);
```

MANIPULATING AN ARRAY

- To remove all empty values use `array_filter()`

```
$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];

$non_empties = array_filter($my_array);
// $non_empties will contain
[1,2,3,4,5,6,7,8];
```

- We can filter with callback:

```
$my_array = [1,2,3,4,5,6,7,8];

$even_numbers = array_filter($my_array,
function($number) {

    return $number % 2 === 0;

});
```

MANIPULATING AN ARRAY

- `array_flip()` function will exchange all keys with its elements.

```
$colors = array(
    'one' => 'red',
    'two' => 'blue',
    'three' => 'yellow',
);
array_flip($colors); //will output
array(
    'red' => 'one',
    'blue' => 'two',
    'yellow' => 'three'
)
```

MANIPULATING AN ARRAY

- When you want to allow only certain keys in your arrays, especially when the array comes from request parameters, you can use `array_intersect_key()` together with `array_flip()`

```
$parameters = ['foo' => 'bar', 'bar' =>
'baz', 'boo' => 'bam'];

$allowedKeys = ['foo', 'bar'];

$fP = array_intersect_key($parameters,
array_flip($allowedKeys));


// $fP contains ['foo' => 'bar', 'bar' =>
'baz]
```

MANIPULATING AN ARRAY

- `array_reduce()` reduces array into a single value. Basically, The `array_reduce()` will go through every item with the result from last iteration and produce new value to the next iteration.

```
$result = array_reduce([1, 2, 3, 4, 5],
function($carry, $item){

    return $carry + $item;

});// Summation of the array

$result = array_reduce([10, 23, 211, 34, 25],
function($carry, $item){

        return $item > $carry ? $item :
$carry;

}); //Get largest number in the array
```

COUNTING ELEMENTS IN AN ARRAY

- We used `count()` before to count the number of elements in an array.
- The function `sizeof()` serves exactly the same purpose.
- The `array_count_values()` this function counts how many times each unique value occurs in the array named `$array`.

COUNTING ELEMENTS IN AN ARRAY

- The function returns an associative array containing a frequency table. This array contains all the unique values from $array as keys. Each key has a numeric value that tells you how many times the corresponding key occurs in $array.
- For example, the code

```
$array = array(4, 5, 1, 2, 3, 1, 2, 1);
$ac = array_count_values($array);
```

COUNTING ELEMENTS IN AN ARRAY

- This creates an array called $ac that contains

Key	Value
4	1
5	1
1	3
2	2
3	1

- This result indicates that 4, 5, and 3 occurred once in $array, 1 occurred three times, and 2 occurred twice.

CONVERTING ARRAYS TO SCALARS

- If you have a non-numerically indexed array with a number of key value pairs, you can turn them into a set of scalar variables using the function `extract`().
- The prototype for `extract()` is as follows:

```
extract(array var_array [, int
extract_type] [, string prefix] );
```

CONVERTING ARRAYS TO SCALARS

- The purpose of `extract`() is to take an array and create scalar variables with the names of the keys in the array. The values of these variables are set to the values in the array. Here is a simple example:

```
$array = array( 'key1' => 'value1', 'key2' => 'value2', 'key3' => 'value3');

extract($array);

echo "$key1 $key2 $key3";
```

- This code produces the following output :

```
value1 value2 value3
```

CONVERTING ARRAYS TO SCALARS

- To convert an indexed array into scalar variables we need to provide the variables as the following:

```
info = array('coffee', 'brown', 'caffeine');


// Listing all the variables

list($drink, $color, $power) = $info;

echo "$drink is $color and $power makes it special.\n";
```

CONTENTS-DAY03

- String Manipulation
- Regular Expressions
- Uploading Files
- Session
- Cookies

FORMATING A STRING FOR DISPLAY

- To trim any excess whitespace from the string:

`chop()`, `rtrim()`, `ltrim()`, **and** `trim()`

```
$name = trim($_POST['name']);
```

- The trim() function strips whitespace from the start and end of a string and returns the resulting string. The characters it strips by default are newlines and carriage returns (\n and \r), horizontal and vertical tabs (\t and \x0B), end-of-string characters (\0), and spaces.
- `chop()` is an alias of `rtrim()`.

FORMATING A STRING FOR DISPLAY

- `nl2br()` function takes a string as a parameter and replaces all the newlines in it with the XHTML
 tag.
- You can apply some more sophisticated formatting using the functions `printf()` and `sprintf()`.

```
string sprintf (string format [, mixed
args...])
```

```
void printf (string format [, mixed args...])
```

CHANGING THE CASE OF A STRING

- `Strtoupper()`, `strtolower()`, `ucfirst()` and `ucwords()`.
- `ucfirst()` Capitalizes first character of string if it's alphabetic.
- `ucwords()` Capitalizes first character of each word in the string that begins with an alphabetic character.

FORMATTING STRINGS FOR STORAGE

- Certain characters are perfectly valid as part of a string but can cause problems, particularly when you are inserting data into a database because the database could interpret these characters as control characters. The problematic ones are quotation marks (single and double), backslashes (\), and the NULL character. You need to escape that using:

`addslashes()` and `stripslashes()`

JOINING AND SPLITTING STRINGS

- Using `explode()`, `implode()`, and `join()`

```
array explode(string separator, string input
[, int limit]);
```

- This function takes a string input and splits it into pieces on a specified separator string. The pieces are returned in an array. You can limit the number of pieces with the optional ***limit*** parameter.
- You can reverse the effects of explode() by using either `implode()` or `join()` .

JOINING AND SPLITTING STRINGS

- Using `strtok()`

```
string strtok(string input, string
separator);
```

- Unlike `explode()`, which breaks a string into all its pieces at one time, `strtok()` gets pieces (called tokens) from a string one at a time.

```
$token = strtok($feedback, " ");

echo $token."<br />";

while ($token != "") {

$token = strtok(" ");

echo $token."<br />";}
```

JOINING AND SPLITTING STRINGS

- Using `substr()`

```
string substr(string string, int start[, int length] );
```

```
$test = 'Your customer service is excellent';
```

```
substr($test, 1);
```

returns our customer service is excellent.

```
substr($test, -9);
```

returns excellent.

```
substr($test, 0, 4);
```

returns the first four characters of the string—namely, Your

COMPARING STRINGS

- Using : `strcmp(), strcasecmp().`

```
int strcmp(string str1, string str2);
```

Returns < 0 if str1 is less than str2; > 0 if str1 is greater than str2, and 0 if they are equal.

```
$string = 'abcdef';
echo $string[0];                    // a
echo $string[3];                    // d
```

GET STRING LENGTH

- You can check the length of a string by using the `strlen()` function. If you pass it a string, this function will return its length. For example, the result of code is 5:

```
echo strlen("hello");
```

SEARCHING STRING

- `strstr()`, `strchr()`, and `stristr()`
- can be used to find a string or character match within a longer string the `strchr()` function is exactly the same as `strstr()`

```
string strstr(string haystack, string needle, [, bool $before_needle = false ] );
```

- You pass the function a haystack to be searched and a needle to be found. If an exact match of the needle is found, the function returns the haystack from the needle onward; otherwise, it returns false.

FINDING THE POSITION OF A SUBSTRING

- `strpos()`, `strrpos()` and `stripos()`
- They return the numerical position of a needle within a haystack.

```
int strpos(string haystack, string needle,
int [offset] );
```

FINDING THE POSITION OF A SUBSTRING

- For example, the following code echoes the value 4 to the browser:

```
$test = "Hello world";

echo strpos($test, "o");
```

- The optional offset parameter specifies a point within the haystack to start searching For example,

```
echo strpos($test, 'o', 5); // 7
```

REPLACING SUBSTRINGS

- `str_replace()` and `substr_replace()`

```
string substr_replace(string string, string
replacement,

int start, int [length] );
```

- This function replaces part of the string string with the string replacement. Which part is replaced depends on the values of the start and optional length parameters.

STRING MANIPULATION

```
int ord ( string $string );

string md5 ( string $str);

string str_repeat ( string $input , int
$multiplier );

string str_shuffle ( string $str );
```

REGULAR EXPRESSIONS

- PHP supports two styles of regular expression syntax: POSIX and Perl.
- A regular expression is a way of describing a pattern in a piece of text.
- we will cover the simpler POSIX style here.

WILD CHARACTER

. (DOT) matches single character.

ex .at will match cat rat hat … or #at

to match a character class we use

[a-z]

ex [a-z]at

will match letters only as hat rat cat NOT #at or $at

CHARACTER CLASSES

- Get all words that contain any vowel

[aeiou]

[a-zA-Z]

[^a-z] Not contain

[a-z]* can be repeated zero or more times.

[a-z]+ can be repeated one or more times.

(very)*large

(very){1, 3}

^[A-Z] beginning with

[A-Z]$ Ending by

cat|rat|hat OR

CHARACTER CLASSES

Class	Matches
`[[:alnum:]]`	Alphanumeric characters
`[[:alpha:]]`	Alphabetic characters
`[[:lower:]]`	Lowercase letters
`[[:upper:]]`	Uppercase letters
`[[:digit:]]`	Decimal digits
`[[:xdigit:]]`	Hexadecimal digits
`[[:punct:]]`	Punctuation
`[[:blank:]]`	Tabs and spaces
`[[:space:]]`	Whitespace characters
`[[:cntrl:]]`	Control characters
`[[:print:]]`	All printable characters
`[[:graph:]]`	All printable characters except for space

CHARACTERS OUTSIDE THE SQURE BRACKETS

Character	Meaning
\	Escape character
^	Match at start of string
$	Match at end of string
.	Match any character except newline (\n)
|	Start of alternative branch (read as OR)
(	Start subpattern
)	End subpattern
*	Repeat zero or more times
+	Repeat one or more times
{	Start min/max quantifier
}	End min/max quantifier
?	Mark a subpattern as optional

CHARACTERS OUTSIDE THE SQURE BRACKETS

Character	Meaning
\	Escape character
^	NOT, only if used in initial position
–	Used to specify character ranges

FINDING SUBSTRINGS WITH REGEX

- `preg_match()` function can be used to parse string using regular expression. The parts of expression enclosed in parenthesis are called subpatterns and with them you can pick individual parts of the string.

```
$str = "<a href=\"http://example.org\">My
Link</a>";

$pattern = "/<a href=\"(.*)\">(.*)<\/a>/";

$result = preg_match($pattern, $str,
$matches);

if($result === 1) {

    // The string matches the expression

    print_r($matches);

}
```

FINDING SUBSTRINGS WITH REGEX

- `preg_match_all()` returns all matching results in the subject string (in contrast to `preg_match()`, which only returns the first one).
- To split a string using REGEX use: `preg_split()`

```
$string = "0| PHP 1| CSS 2| HTML 3| AJAX 4| JSON";

$array = preg_split("/[0-9]+\|/", $string);
```

UPLOADING FILES

```
<form action="upload.php" method="post"
enctype="multipart/form-data"/>

<div>

<input type="hidden" name="MAX_FILE_SIZE"
value="1000000" />

<label for="userfile">Upload a file:</label>

<input type="file" name="userfile"
id="userfile"/>

<input type="submit" value="Send File"/>

</div>

</form>
```

UPLOADING FILES

- In the <form> tag, you must set the attribute `enctype="multipart/form-data"` to let the server know that a file is coming along with the regular information.
- You may have a form field that sets the maximum size file that can be uploaded. This is a hidden field and is shown here as

```
<input type="hidden" name="MAX_FILE_SIZE"
value=" 1000000">
```

- Note that the `MAX_FILE_SIZE` form field is optional, as this value can also be set server-side in bytes.

UPLOADING FILES

- The data you need to handle in your PHP script is stored in the superglobal array `$_FILES`.
- `$_FILES['userfile']['tmp_name']` is the place where the file has been temporarily stored on the web server.
- `$_FILES['userfile']['name']` is the file's name on the user's system.
- `$_FILES['userfile']['size']` is the size of the file in bytes.
- `$_FILES['userfile']['type']` is the MIME type of the file—for example, text/plain or image/gif.
- `$_FILES['userfile']['error']` will give you any error codes associated with the file upload .

UPLOADING FILES

```
if ($_FILES[‘userfile’][‘error’] > 0)
{
        echo ‘Problem: ‘;
        switch ($_FILES[‘userfile’][‘error’])
        {
        case 1: echo ‘File exceeded upload_max_filesize’;
        break;
        case 2: echo ‘File exceeded max_file_size’;
        break;
        case 3: echo ‘File only partially uploaded’;
        break;
        case 4: echo ‘No file uploaded’;
        break;
        case 6: echo 'Cannot upload file: No temp directory specified';
        break;
        case 7: echo 'Upload failed: Cannot write to disk';
        break;
        }
        exit;
}
```

UPLOADING FILES

```
// put the file where we'd like it
$upfile = '/uploads/'.$_FILES['userfile']['name'] ;
// Does the file have the right MIME type?
if ($_FILES['userfile']['type'] != 'text/plain')
{
echo 'Problem: file is not plain text';
exit;
}
if (is_uploaded_file($_FILES['userfile']['tmp_name']))
{
        if (!move_uploaded_file($_FILES['userfile']['tmp_name'], $upfile))
        {
        echo 'Problem: Could not move file to destination directory';
        exit; }
else
{
echo 'Problem: Possible file upload attack. Filename: ';
echo $_FILES['userfile']['name'];
exit;
}
echo 'File uploaded successfully<br><br>';
```

UPLOADING FILES

File Upload Configuration Settings in php.ini

Directive	Description	Default Value
`file_uploads`	Controls whether HTTP file uploads are allowed. Values are On or Off.	On
`upload_tmp_dir`	Indicates the directory where uploaded files will temporarily be stored while they are waiting to be processed. If this value is not set, the system default will be used.	NULL
`upload_max_filesize`	Controls the maximum allowed size for uploaded files. If a file is larger than this value, PHP will write a 0 byte placeholder file instead.	2M
`post_max_size`	Controls the maximum size of POST data that PHP will accept. This value must be greater than the value for the `upload_max_filesize` directive, since it is the size for all of the post data, including any files to be uploaded.	8M

SESSION CONTROL

- ***HTTP is a stateless protocol***. This means that the protocol has no built-in way of maintaining state between two transactions. When a user requests one page, followed by another, HTTP does not provide a way for you to tell that both requests came from the same user.
- The idea of session control is to be able to track a user during a single session on a website. If you can do this, you can easily support logging in a user and showing content according to her authorization level or personal preferences.

SESSION CONTROL

- Sessions in PHP are driven by a **unique session ID**, a cryptographically random number.
- This session ID is generated by PHP and stored on the server side for the lifetime of a session.
- It can be either stored on a user's computer in a **cookie** or passed along through URLs.
- **Cookies** are a different solution to the problem of preserving state across a number of transactions while still having a clean-looking URL.

SESSION CONTROL

- A ***cookie*** is a small piece of information that scripts can store on a client-side machine.
- You can manually set cookies in PHP using the setcookie() function. It has the following prototype:

```
bool setcookie (string name [, string value
[, int expire [, string path[, string domain
[, int secure]]]]])
```

SESSION CONTROL

- If you set a cookie as

```
setcookie ('mycookie', 'value');
```

- you will can access the cookie via `$_COOKIE['mycookie'];`
- You can delete a cookie by calling `setcookie()` again with the same cookie name and an expiry time in the past.

SESSION CONTROL

- You can use a dual cookie/URL method:

```
session_set_cookie_params($lifetime, $path,
$domain [, $secure]);
```

- to see the contents of the cookie set by session control:

```
Array session_get_cookie_params();
```

SESSION CONTROL

- You can use a dual cookie/URL method:

```
session_set_cookie_params($lifetime, $path,
$domain [, $secure]);
```

- to see the contents of the cookie set by session control:

```
Array session_get_cookie_params();
```

- The other method it can use is adding the **session ID** to the URL. You can set this to happen automatically if you set the `session.use_trans_sid` directive in the php.ini file.

SESSION CONTROL

- The basic steps of using sessions are

1. Starting a session
2. Registering session variables
3. Using session variables
4. Deregistering variables and destroying the session

SESSION CONTROL

- Before you can use session functionality, you need to actually begin a session.

```
session_start();
```

- It's essential to call `session_start()` at the start of all your scripts that use sessions. If this function is not called, anything stored in the session will not be available to this script.
- you can begin a session is to set PHP to start one automatically when someone comes to your site. You can do this by using the `session.auto_start` option in your php.ini file.

SESSION CONTROL

- To create a session variable, you simply set an element in this array, as follows:

```
$_SESSION['myvar'] = 5;
```

- When you are finished with a session variable, you can unset it. You can do this directly by unsetting the appropriate element of the `$_SESSION` array, as in this example:

```
unset($_SESSION['myvar']);
```

SESSION CONTROL

- You should not try to unset the whole `$_SESSION` array because doing so will effectively disable sessions. To unset all the session variables at once, use

```
$_SESSION = array();
```

- When you are finished with a session, you should first unset all the variables and then call

```
session_destroy();
```

- to clean up the session ID.

CONTENTS-DAY04

- Accessing MySQL Using mysqli
- PDO
- Date and Time Functions

USING mysqli

- The mysqli interface is an improvement (it means "MySQL Improvement extension") of the mysql interface, which was deprecated in version 5.5 and is removed in version 7.0.
- The mysqli extension, or as it is sometimes known, the MySQL improved extension, was developed to take advantage of new features found in MySQL systems versions 4.1.3 and newer.
- It features a dual interface: the older, procedural style and a new, object-oriented programming (OOP) style. The deprecated mysql had only a procedural interface

START THE CONNECTION

- To connect to the Database we use one of these ways:

Procedural

```
$conn = mysqli_connect("localhost","user","password","db");
```

Object Oriented

```
$conn = new mysqli("localhost","user","password","db");
```

- We can omit the database and use `mysqli_select_db(db_resource, db_name);`

CHECKING FOR ERRORS

- To connect to the Database we use one of these ways:

Procedural

```
if (mysqli_connect_errno()) {
    trigger_error(mysqli_connect_error());
}
```

Object Oriented

```
if ($conn->connect_errno) {
    trigger_error($db->connect_error);
}
```

ESCAPING STRINGS

- Escaping strings is an older (and less secure) method of securing data for insertion into a query:

Procedural

```
$escaped = mysqli_real_escape_string($conn, $_GET['var']);
```

Object Oriented

```
$escaped = $conn->real_escape_string($_GET['var']);
```

- This method does not provide a full secure query. Consider this:

```
$id = mysqli_real_escape_string("1 OR 1=1");
```

- As you may notice that MySQL's escaping function is **designed to make data comply with SQL syntax**. It's **NOT** designed to make sure that MySQL can't confuse user data for SQL instructions.

QUERY EXECUTION

- To execute a query:

Procedural

```
$result = mysqli_query($conn, "SELECT * FROM `people`");
```

Object Oriented

```
$result = $conn->query("SELECT * FROM `people`");
```

- If the query is OK, it will return mysqli_stmt object, otherwise it will return false

ROWSET COUNT

- Get the number of rows returned:

Procedural

```
$rows_count = mysqli_num_rows($result);
```

Object Oriented

```
$rows_count = $result->num_rows;
```

- We can use $rows_count in a for loop.

LOOPING ON ROWSET

- We can loop on the rowset and fetch one row each time:

Procedural

```
while($row = mysqli_fetch_assoc($result)) {
    var_dump($row);
}
```

Object Oriented

```
while($row = $result->fetch_assoc()) {
    var_dump($row);
}
```

- Several variations can be used to get results from a result identifier. Instead of an array with named keys, you can retrieve the results in an enumerated array with `mysqli_fetch_row()`, You could also fetch a row into an object with the `mysqli_fetch_object()` function.

MODIFYING DATA

- We can execute also insert, update and delete statement:

Procedural

```
$query = "insert into users values ('".$username."',
'".$password."')";
mysqli_query($conn, $query );
echo mysqli_affected_rows();
```

Object Oriented

```
$query = "insert into users values ('".$username."',
'".$password."')";
$conn->query($query);
echo $conn->affected_rows;
```

- To retrieve the last inserted id we can use `$conn->insert_id` or `mysqli_insert_id($conn)`

CLOSING THE CONNECTION

- We loop on the rowset and fetch one row each time:

Procedural

```
mysqli_free_result($result);
mysqli_close($conn);
```

Object Oriented

```
$result->free();
$conn->close();
```

DEBUGGING IN mysqli

- If a query has failed, we get the error by:

Procedural

```
echo mysqli_error($conn);
```

Object Oriented

```
echo $conn->error;
```

PREPARED STATEMENTS

- The basic concept of a prepared statement is that you send a template of the query you want to execute to MySQL and then send the data separately.
- You can send multiple lots of the same data to the same prepared statement; this capability is particularly useful for bulk inserts.
- They are useful for speeding up execution when you are performing large numbers of the same query with different data
- Prepared statements are the recommended solution for the prevention of SQL injection.

PREPARED STATEMENTS

- We can execute also insert, update and delete statement:

Procedural

```
$sql = "insert into books values(?, ?)";
if ($stmt = mysqli_prepare($conn, $sql)) {
  mysqli_stmt_bind_param($stmt, "si", $title, $price);
  $result = mysqli_stmt_execute($stmt);
  // Fetch data here
  mysqli_stmt_close($stmt);
}
```

Object Oriented

```
$sql = "insert into books values(?, ?)";
if ($stmt = $conn->prepare($sql)) {
  $stmt->bind_param("si", $title, $price);
  $result = $stmt->execute();
 // Fetch data here
$stmt->close();
}
```

PREPARED STATEMENTS

- We have to pass the data type of the parameters to bind_param function, the type is one of these types:

Character	Description
i	corresponding variable has type integer
d	corresponding variable has type double
s	corresponding variable has type string
b	corresponding variable has type binary

PDO

- The PDO (PHP Data Objects) extension allows developers to connect to numerous different types of databases and execute queries against them in a uniform, object oriented manner.
- PDO provides a *data-access abstraction* layer, which means that, regardless of which database you're using, you use the same functions to issue queries and fetch data.
- PDO supports a wide range of databases such as
  - MS SQL Server
  - Informix
  - Oracle
  - MySQL
  - PostgreSQL
  - SQLite

STARTING A CONNECTION

- To start a connection, we need to prepare Data Source Name (DSN).

```
$dsn = "mysql:host=localhost;dbname=testdb";
$db = new PDO($dsn, $username, $password);
// setup PDO to throw an exception if an invalid query is provided
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$query = "SELECT * FROM users WHERE class = ?";
$statement = $db->prepare($query);
$parameters = [ "221B" ];
$statement->execute($parameters);
while ($row = $statement->fetch(PDO::FETCH_ASSOC)) {
    do_stuff($row);
}
```

PREPARED STATEMENTS PLACEHOLDERS

- PDO supports two kinds of placeholders:

1. Named placeholders. A colon(:), followed by a distinct name (eg. :user)

```
$sql = 'SELECT name, email, user_level FROM
users WHERE userID = :user';

$prep = $conn->prepare($sql);

$prep->execute([':user' => $_GET['user']]);

 $result = $prep->fetchAll();
```

PREPARED STATEMENTS PLACEHOLDERS

2. Traditional SQL positional placeholders, represented as ?

```
$sql = 'SELECT name, user_level FROM users
WHERE userID = ? AND user_level = ?';

$prep = $conn->prepare($sql);

$prep->execute([$_GET['user'],
$_GET['user_level']]);

$result = $prep->fetchAll();
```

DATABASE TRANSACTIONS WITH PDO

- PDO provides simple methods for beginning, committing, and rollbacking back transactions:

```
try {
    $statement = $pdo->prepare("UPDATE user SET name =
:name");
    $pdo->beginTransaction();
    $statement->execute([":name"=>'Islam']);
    $statement->execute([":name"=>'Ahmed']);
    $pdo->commit();
} catch (\Exception $e) {
    if ($pdo->inTransaction()) {
        $pdo->rollback();
    }
    throw $e;
}
```

USEFUL OPERATIONS

- To get the number of affected rows by a query:

```
$query = $db->query("DELETE FROM user WHERE
name = 'Ahmed'");

$count = $query->rowCount();
```

- To get the last inserted id:

```
$query = "INSERT INTO user(username, email)
VALUES ('Islam', 'isalah@iti.gov.eg')";

$pdo->query($query);

$id = $pdo->lastInsertId();
```

DISPLAYING FORMATED DATE

- `date()` function takes two parameters, one of them optional.
- The first one is a format string, and the second, optional one is a Unix timestamp. If you don't specify a timestamp, `date()` will default to the current date and time.

```
date('jS F Y');
```

FORMAT STRING

Code	Description
a	Morning or afternoon, represented as two lowercase characters, either `am` or `pm`..
A	Morning or afternoon, represented as two uppercase characters, either `AM` or `PM`.
B	Swatch Internet time, a universal time scheme. More information is available at http://www.swatch.com/.
c	ISO 8601 date. A date is represented as YYYY-MM-DD. An uppercase `T` separates the date from the time. The time is represented as HH:MM:SS. Finally, the time zone is represented as an offset from Greenwich mean time (GMT)—for example, `2008-06-26T21:04:42+11:00`. (This format code was added in PHP5.)
d	Day of the month as a two-digit number with a leading zero. The range is from `01` to `31`.
D	Day of the week in three-character abbreviated text format. The range is from `Mon` to `Sun`.
e	Timezone identifier (added in PHP 5.1.0)
F	Month of the year in full text format. The range is from `January` to `December`.

FORMAT STRING

g	Hour of the day in 12-hour format without leading zeros. The range is from 1 to 12.
G	Hour of the day in 24-hour format without leading zeros. The range is from 0 to 23.
h	Hour of the day in 12-hour format with leading zeros. The range is from 01 to 12.
H	Hour of the day in 24-hour format with leading zeros. The range is from 00 to 23.
i	Minutes past the hour with leading zeros. The range is from 00 to 59.
I	Daylight savings time, represented as a Boolean value. This format code returns 1 if the date is in daylight savings and 0 if it is not.
j	Day of the month as a number without leading zeros. The range is from 1 to 31.
l	Day of the week in full-text format. The range is from Sunday to Saturday.
L	Leap year, represented as a Boolean value. This format code returns 1 if the date is in a leap year and 0 if it is not.
m	Month of the year as a two-digit number with leading zeros. The range is from 01 to 12.
M	Month of the year in three-character abbreviated text format. The range is from Jan to Dec.
n	Month of the year as a number without leading zeros. The range is from 1 to 12.

FORMAT STRING

o	ISO-8601 year number. This has the same value as Y, except that if the ISO week number (W) belongs to the previous or next year, the year is used instead (added in PHP 5.1.0).
O	Difference between the current time zone and GMT in hours—for example, +1600.
r	RFC822-formatted date and time—for example, Wed, 1 Jul 2008 18:45:30 +1600. (This code was added in PHP 4.0.4.)
s	Seconds past the minute with leading zeros. The range is from 00 to 59.
S	Ordinal suffix for dates in two-character format. It can be st, nd, rd, or th, depending on the number it follows.
t	Total number of days in the date's month. The range is from 28 to 31.
T	Time zone setting of the server in three-character format—for example, EST.
U	Total number of seconds from January 1, 1970, to this time; also known as a *Unix timestamp* for this date.
w	Day of the week as a single digit. The range is from 0 (Sunday) to 6 (Saturday).
W	Week number in the year; ISO-8601 compliant. (This format code was added at PHP 4.1.0.)
y	Year in two-digit format—for example, 08.
Y	Year in four-digit format—for example, 2008.
z	Day of the year as a number. The range is 0 to 365.
Z	Offset for the current time zone in seconds. The range is -43200 to 43200.

DEALING WITH UNIX TIMESTAMPS

- Unix systems store the current time and date as a 32-bit integer containing the number of seconds since midnight, January 1, 1970, GMT.
- They do have similar problems, though, because they can represent only a limited span of time using a 32-bit integer. If your software needs to deal with events before 1902 or after 2038, you will be in trouble.

DEALING WITH UNIX TIMESTAMPS

- If you want to convert a date and time *to* a Unix timestamp, you can use the `mktime()` function. It has the following prototype:

```
int mktime ([int hour[, int minute[, int
second[, int month[,

int day[, int year [, int is_dst]]]]]]);

$timestamp = mktime();

$timestamp = time();

$timestamp = date("U");
```

USING `getdate()`

- Another date-determining function you might find useful is `getdate()`.This function has the following prototype:

```
array getdate ([int timestamp]);
```

- It takes an optional timestamp as a parameter and returns an array representing the parts of that date and time.

USING getdate()

Key	Value
seconds	Seconds, numeric
minutes	Minutes, numeric
hours	Hours, numeric
mday	Day of the month, numeric
wday	Day of the week, numeric
mon	Month, numeric
year	Year, numeric
yday	Day of the year, numeric
weekday	Day of the week, full-text format
month	Month, full-text format
0	Timestamp, numeric

VALIDATING DATES

- You can use the `checkdate()` function to check whether a date is valid. This capability is especially useful for checking user input dates. The `checkdate()` function prototype:

```
int checkdate (int month, int day, int year);
```

- For example,

```
checkdate(2, 29, 2008);
```

- returns true, whereas

```
checkdate(2, 29, 2007);
```

- does not.

FORMATTING TIMESTAMPS

- You can format a timestamp according to the system's locale (the web server's local settings) using the strftime() function.This function has the following prototype:

```
string strftime ( string $format [, int $timestamp] )
```

FORMATTING TIMESTAMPS

- As Example:

```
<?php
echo strftime('%A<br />');
echo strftime('%x<br />');
echo strftime('%c<br />');
echo strftime('%Y<br />');
?>
```

- displays the current system timestamp in four different formats. This code will produce output similar to the following:

Friday

03/16/07

03/16/07 21:17:24

2007

FORMATTING TIMESTAMPS

Code	Description
%a	Day of week (abbreviated)
%A	Day of week
%b *or* %h	Month (abbreviated)
%B	Month
%c	Date and time in standard format
%C	Century
%d	Day of month (from 01 to 31)
%D	Date in abbreviated format (mm/dd/yy)
%e	Day of month as a two-character string (from ' 1' to '31')
%g	Year according to the week number, two digits
%G	Year according to the week number, four digits
%H	Hour (from 00 to 23)
%I	Hour (from 1 to 12)

FORMATTING TIMESTAMPS

%j	Day of year (from 001 to 366)
%m	Month (from 01 to 12)
%M	Minute (from 00 to 59)
%n	Newline (\n)
%p	am or pm (or local equivalent)
%r	Time using a.m./p.m. notation
%R	Time using 24-hour notation
%S	Seconds (from 00 to 59)
%t	Tab (\t)
%T	Time in hh:ss:mm format

FORMATTING TIMESTAMPS

%u	Day of week (from 1 – Monday to 7 – Sunday)
%U	Week number (with the first Sunday of the year being the first day of the first week)
%V	Week number (with the first week in the year with at least four days counting as week number 1)
%w	Day of week (from 0 – Sunday to 6 – Saturday)
%W	Week number (with the first Monday of the year being the first day of the first week)
%x	Date in standard format (without the time)
%X	Time in standard format (without the date)
%y	Year (two digits)
%Y	Year (four digits)
%z *or* %Z	Timezone

MYSQL DATES

- Dates and times in MySQL are handled in ISO 8601 format. Times work relatively intuitively, but ISO 8601 requires you to enter dates with the year first.
- For example, you could enter March 29, 2008, either as 2008-03-29 or as 08-03-29.
- from PHP, you can easily put them into the correct format by using the date() function.

CALCULATING DATES IN PHP

- A simple way to work out the length of time between two dates in PHP is to use the difference between Unix timestamps.

```
$bdayunix = mktime (0, 0, 0, $month, $day, $year);

$nowunix = time(); // get unix ts for today

$ageunix = $nowunix - $bdayunix;
```

- This is not a timestamp but instead the age of the person measured in seconds

```
$age = floor($ageunix / (365 * 24 * 60 * 60));
```