69/B Monwara Plaza (Level-5)
East Panthapath
Dhaka-1205
Cell: 01713 432003

Lecture Sheet on "PHP+MySQL"
Day-6: (PHP)

Objectives	**Topics**
❖ Create PHP scripts using multiple files ❖ Create and call functions ❖ Create functions that take arguments and return values ❖ Understand date time functions	❖ Including files ❖ Creating your own functions ❖ Creating a function that takes arguments ❖ Returning values from a function ❖ Date and time functions

include(): The include() statement includes and evaluates the specified file.
When a file is included, the code it contains inherits the variable scope of the line on which the include occurs. Any variables available at that line in the calling file will be available within the called file, from that point forward. However, all functions and classes defined in the included file have the global scope.
Example: Basic include() example
vars.php

```
<?php
        $color = 'green';
        $fruit = 'apple';
?>
```

test.php

```
<?php
        echo "A $color $fruit"; // A
        include 'vars.php';
        echo "A $color $fruit"; // A green apple
?>
```

If the include occurs inside a function within the calling file, then all of the code contained in the called file will behave as though it had been defined inside that function. So, it will follow the variable scope of that function. An exception to this rule are magic constants which are evaluated by the parser before the include occurs.
Example: Including within functions

```
<?php
function foo()
{
   global $color;
   include 'vars.php';
   echo "A $color $fruit";
}
/* vars.php is in the scope of foo() so     *
 * $fruit is NOT available outside of this  *
 * scope.  $color is because we declared it *
 * as global.                               */
foo();                    // A green apple
echo "A $color $fruit";   // A green ?>
```

The **include_once()** statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the include() statement, with the only difference being that if the code from a file has already been included, it will not be included again. As the name suggests, it will be included just once.
include_once() should be used in cases where the same file might be included and evaluated more than once during a particular execution of a script, and you want to be sure that it is included exactly once to avoid problems with function redefinitions, variable value reassignments, etc.
Return values are the same as with include(). If the file was already included, this function returns TRUE
Note: include_once() was added in PHP 4.0.1
Note: Be aware, that the behaviour of include_once() and require_once() may not be what you expect on a non case sensitive operating system (such as Windows).
Example: include_once() is case insensitive on Windows

```
<?php
        include_once "a.php"; // this will include a.php
        include_once "A.php"; // this will include a.php again on Windows! (PHP 4 only)
?>
```

require() and include() are identical in every way except how they handle failure. They both produce a Warning, but require() results in a Fatal Error. In other words, don't hesitate to use require() if you want a missing file to halt processing of the page. include() does not behave this way, the script will continue regardless. Be sure to have an appropriate include_path setting as well.
Example: Basic require() examples

```
<?php
        require 'prepend.php';
        require $somefile;
        require ('somefile.txt');
?>
```

User-defined functions
Any valid PHP code may appear inside a function, even other functions and class definitions. Function names follow the same rules as other labels in PHP. A valid function name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus: [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]. When a function is defined in a conditional manner, its definition must be processed prior to being called.
Example: Conditional functions

```
<?php
$makefoo = true;
/* We can't call foo() from here
   since it doesn't exist yet,
   but we can call bar() */
bar();
if ($makefoo) {
  function foo()
  {
    echo "I don't exist until program execution reaches me.\n";
  }
}
/* Now we can safely call foo()
   since $makefoo evaluated to true */
if ($makefoo) foo();
function bar()
{
  echo "I exist immediately upon program start.\n";
}
?>
```

Example: Functions within functions

```
<?php
function foo()
{
  function bar()
  {
    echo "I don't exist until foo() is called.\n";
  }
}
/* We can't call bar() yet
   since it doesn't exist. */
foo();
/* Now we can call bar(),   foo()'s processesing has   made it accessible. */
bar();
?>
```

All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa.

PHP does not support function overloading, nor is it possible to undefine or redefine previously-declared functions.

Note: Function names are case-insensitive, though it is usually good form to call functions as they appear in their declaration.

It is possible to call **recursive functions** in PHP. However avoid recursive function/method calls with over 100-200 recursion levels as it can smash the stack and cause a termination of the current script.

Example: **Recursive functions**

```
<?php
function recursion($a)
{
    if ($a < 20) {
        echo "$a\n";
        recursion($a + 1);
    }
} ?>
```

Date and Time Functions

These functions allow you to get the date and time from the server where your PHP scripts are running. You can use these functions to format the date and time in many different ways.

checkdate — Validate a Gregorian date, [bool **checkdate** (int $month, int $day, int $year)]
date_modify — Alters the timestamp, $date->modify(string $modify
date_sun_info — Returns an array with information about sunset/sunrise and twilight begin/end. [array **date_sun_info** (int $time, float $latitude, float $longitude)]
date_sunrise — Returns time of sunrise for a given day and location, [mixed **date_sunrise** (int $timestamp [, int $format [, float $latitude [, float $longitude [, float $zenith [, float $gmt_offset]]]]])] ,
date_sunset — Returns time of sunset for a given day and location
date_time_set — Sets the time, [**date_time_set** (DateTime $object, int $hour, int $minute [, int $second])],
date_default_timezone_set— Sets the time zone for the DateTime object, [date_default_timezone_set("Asia/Dacca")]
date — Format a local time/date, see the php manual for more reference.
microtime — Return current Unix timestamp with microseconds
mktime — Get Unix timestamp for a date, [**mktime** ([int $hour [, int $minute [, int $second [, int $month [, int $day [, int $year [, int $is_dst]]]]]]])]
strtotime — Parse about any English textual datetime description into a Unix timestamp
time — Return current Unix timestamp
DateTimeZone::listIdentifiers -
date_default_timezone_set — Sets the default timezone used by all date/time functions in a script

Calculate Date difference:

If you want to get the difference between to dates or time values, then you first need to get the values in the same format. The best way is to convert both dates into Unix timestamp format. In this case the date and time information is stored as an integer so we can calculate the difference easily. You can get the Unix timestamp in various ways as you can see below:

```
<?php
    // Get current time
    $date1 = time();
    // Get the timestamp of 2006 October 20
    $date2 = mktime(0,0,0,10,20,2006);
?>
```

Now as you have the values getting the difference is quite easy. However the result will be an integer value, but you probably want to get it in days, hours and minutes. To do so we need to convert back our int value into a usable date format. We know that the difference is in seconds so we can get how many full days it is. To get it we use the following code:

```
<?php
     $dateDiff = $date1 - $date2;
     $fullDays = floor($dateDiff/(60*60*24));
     echo "Differernce is $fullDays days";
?>
```

We used the floor function as we want to get only the complete days. Besides this the 60*60*24 represents that a day has 24 ours and each hour has 60 minutes and each minute has 60 seconds.

As next step we need to get how many hours are still present in the remaining seconds. So from the original difference we need to remove the complete days in seconds and from the remaining value we can calculate the full hours similar to as we calculated the full days. Repeating these steps for minutes as well at the end we get the difference between the original dates. The complete calculation looks like this:

```
<?php
     $dateDiff = $date1 - $date2;
     $fullDays = floor($dateDiff/(60*60*24));
      $fullHours = floor(($dateDiff-($fullDays*60*60*24))/(60*60));
     $fullMinutes = floor(($dateDiff-($fullDays*60*60*24)-($fullHours*60*60))/60);
     echo "Differernce is $fullDays days, $fullHours hours and $fullMinutes minutes.";
?>
```