INTRODUCTION TO PHP

CP 311: Internet programming and applications II

OBJECTIVES

- Basic Features
- PHP Support
- PHP Variables
- Use of Control Structures and Functions
- Setting Up PHP Forms
- Operators
- PHP Files Manipulation

AN EXAMPLE: SIMPLE PHP PROGRAM

```
<html>
  <head>
        <title>Simple PHP Example</title>
  </head>
  <body>
        <?php
              echo "My first PHP script!";
        ?>
  </body>
</html>
```

- An HTML script with embedded *php* code to do something (output text).
- Perl or C – would write a program with lots of commands to output an HTML

PHP: HYPERTEXT PRE-PROCESSOR

A widely-used open source general-purpose scripting language

- Suited for Web development
- Can be embedded into HTML
- Its syntax draws upon C, Java, and Perl

Its main goal is to ease and quicken development

of dynamically generated web pages

FEATURES

- Declaration:
  Start and *end* tags allow to jump into and out of "PHP mode". Three possible forms of declarations

  a)
  ```
  <?
      PHP Code In Here
  ?>
  ```

  b)
  ```
  <?php
      PHP Code In Here
  ?>
  ```

  c)
  ```
  <script language="php">
      PHP Code In Here
  </script>
  ```

- Syntax
  - White space has no effect
  - Statements end with a semicolon
- Running
  - Run from the command line or via a web browser
  - Web server interpretes php code, replaces blocks with output of php command

COMMENTS

All styles of comments are supported

```
//  this is a comment


/* This comment spans multiple lines.
   It is used in the same way as in C/C++ or
  Java
 */
```

PHP VS. CLIENT-SIDE JAVASCRIPT

- Unlike client-side JavaScript, PHP is executed on the server.
  - **PHP concentrates on server-side scripting**
  - **As it is not run on the user's browser, no need to worry about compatibility issues.**
- The client would receive the results of the running script, with no way of determining what the underlying code may be.
- You can even configure your web server to process all your HTML files with PHP

WHAT CAN PHP DO?

- Anything ☺ - haha …
- Like CGI, PHP is focused on server-side scripting, so you can
  - collect form data
  - generate dynamic page content
  - send and receive cookies

PHP can also do much more

WHAT CAN SCRIPTS ADD TO WEB SITES?

Scripting can add '<u>interactive</u>' features

- Simple interactive features
  - feedback forms
  - guestbooks
  - message boards
  - counters
- Advanced features
  - portal systems
  - content management
  - advertising managers
  - creation of shopping carts for e-commerce websites.

SERVER-SIDE SCRIPTING

- To work, server-side scripting needs
  - the PHP parser (CGI or server module)
  - a web server
  - a web browser
- The web server has to have a connected PHP installation.
- The PHP program output is accessed with a web browser, viewing the PHP page through the server.

COMMAND LINE SCRIPTING

- A PHP script can run without any server or browser.
  - Only the PHP parser is needed in this way.
  - Ideal usage for scripts regularly executed using cron (on *nix or Linux) or Task Scheduler (on Windows).
  - Can also be used for simple text processing tasks.

PHP SUPPORT

- PHP is supported by all major operating systems
  - Linux, many Unix variants (including HP-UX, Solaris and OpenBSD), Microsoft Windows, Mac OS X, RISC OS

- PHP has support for most of the web servers
  - Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet servers, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd, and many others.

- For the majority of the servers PHP has a module, for the others supporting the CGI standard, PHP can work as a CGI processor.

PHP SUPPORT (CTD.)

- PHP offers the freedom to choose
  - an operating system and a web server.
  - procedural programming or object oriented programming, or a mixture of them.
  - PHP 5 introduces a *complete object model*.
- PHP can output
  - HTML or any text, such as XHTML or any other XML file
  - images
  - PDF files
  - Flash movies (using libswf and Ming) generated on the fly.
  - can autogenerate these files, and save them in the file system, instead of printing it out, forming a ***server-side cache*** for your dynamic content.

DATABASE SUPPORT

- PHP supports a wide range of databases

  Adabas D, InterBase,
  PostgreSQL, dBase, FrontBase,
  SQLite, Empress, **mSQL**,
  Solid, Sybase, FilePro (read-only)
  Hyperwave, **MySQL**, Direct MS-SQL,
  Velocis, IBM DB2, ODBC,
  Unix dbm, Informix, **Oracle** (OCI7 & OCI8)
  IngresOvrimos

- Writing a database-enabled web page is simple.

OTHER PHP SUPPORTED FEATURES

- Talking to other services using protocols such as LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM
- Support for the WDDX complex data exchange between virtually all Web programming languages.
- Support for instantiation of Java objects and using them transparently as PHP objects.
- Can use php CORBA extension to access remote objects.
- Text processing features, from the POSIX Extended or Perl regular expressions to parsing XML documents.
- While using PHP in the e-commerce field, you'll find the Cybercash payment, CyberMUT, VeriSign Payflow Pro and MCVE functions useful for your online payment programs.
- Other extensions include the *mnoGoSearch* search engine functions, the IRC Gateway functions, many compression utilities (gzip, bz2), calendar conversion, translation...

WEB SERVER WITH PHP SUPPORT

- Default PHP files extension is ***.php***
- PHP is installed in a PHP supported server
  - put php files in your web directory and the server will automatically parse them for you
  - no compilation or installation of extra tools is needed
- PHP-enabled files are HTML files with magical tags that let you do all sorts of things.
- Tools needed are:
  - a web server, such as Apache, and PHP.
  - a database, such as MySQL.

HELLO.PHP SIMPLE PHP EXAMPLE

```
<html>
  <head>
        <title>PHP Example</title>
  </head>
  <body>
      <?php  echo "<p>My first PHP script</p>"; ?>

  </body>
</html>
```

put ***hello.php*** in your web server's root directory (*DOCUMENT_ROOT*)

HELLO.PHP SIMPLE PHP EXAMPLE

Use browser to access the file with web server's URL, ***http://localhost/hello.php*** or ***http://127.0.0.1/hello.php***

This file will be parsed by PHP and the following output will be sent to your browser:

```
<html>
  <head>
      <title>PHP Example</title>
  </head>
  <body>
      <p>My first PHP script</p>
  </body>
</html>
```

GET SYSTEM INFORMATION FROM PHP

The *phpinfo()* function

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

- *phpinfo()* prints out all the information about PHP on your server
- shows useful information about the system and setup such as
  - available predefined variables
  - loaded PHP modules
  - configuration settings
- Note that in PHP lines end with a ***semicolon***

PRINTING TEXT

- We use ***print*** to output text, variables or a combination of the two on the screen.
  - Example

```
<?php
        print("Hello world!"); //displays "Hello world!"
?>
```

  - *Print (a command), Hello world (text to be printed) and the line ends with a semicolon.*
  - *The line is enclosed in standard PHP tags.*

VARIABLES

- Represented by a dollar sign followed by the name of the variable.
- They are case-sensitive
- Starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed as: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

VARIABLE DEFINITIONS

```
<?php
  $FName  = ‘Siphael';
  $LName = ‘Betuel';
  $Hello     = 'Hello, Im PHP novice!';
  echo  $FName, $LName;          // outputs “Siphael, Betuel"
  print($Hello);                 // output “Hello World, Im PHP novice!”
  // $4site = 'not yet';         // invalid; starts with a number
  $_4site = 'not yet';           // valid; starts with an underscore
  $gävle = 'whatever';           // valid; 'ä' is extended ASCII 228.
?>
```

FORMATTING OUTPUTS

```
<?php
        $WelcomeText = "Hello, welcome to my
website.";
        print($WelcomeText);
?>
```

Outputs: Hello, welcome to my website.

By default, everything is just output in the browser's default font.

- As PHP is a server side, the code is executed before the page is sent to the browser.
- only the resulting information from the script is sent to the browser

FORMATTING OUTPUTS (CTD.)

- To format text using HTML.
  - include standard HTML markup in your scripts and strings

```
<?php
        print("<font face=\"Arial\" color\"#FF0000\">Hello and
welcome to my website.</font>");
?>
```

Outputs: Hello and welcome to my website.

  - Only this code is sent to the browser:

```
<font face="Arial" color="#FF0000">Hello and welcome to my
website.</font>
```

  - Note that quotations in the HTML code are ignored by the PHP by preceeding them with a slash character as shown here: \"Arial\"

PHP GLOBAL VARIABLES - SUPERGLOBALS

- Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

PHP GLOBAL VARIABLES - SUPERGLOBALS

- The PHP superglobal variables are:
- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

PHP GLOBAL VARIABLES - SUPERGLOBALS

- PHP $GLOBALS
- $GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).
- PHP stores all global variables in an array called $GLOBALS[*index*].
- The *index* holds the name of the variable.

PHP GLOBAL VARIABLES - SUPERGLOBALS

- PHP $_SERVER
- $_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.
- The example below shows how to use some of the elements in $_SERVER:

PHP GLOBAL VARIABLES - SUPERGLOBALS

```php
<?php
 echo $_SERVER['PHP_SELF'];
 echo $_SERVER['SERVER_NAME'];
 echo $_SERVER['HTTP_HOST'];
 echo $_SERVER['HTTP_REFERER'];
 echo $_SERVER['HTTP_USER_AGENT'];
 echo $_SERVER['SCRIPT_NAME'];
 ?>
```

PHP GLOBAL VARIABLES - SUPERGLOBALS

$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['SERVER_NAME']	Returns the name of the host server
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol

PHP GLOBAL VARIABLES - SUPERGLOBALS

PHP $_REQUEST

PHP $_REQUEST is a PHP super global variable which is used to collect data after submitting an HTML form.

PHP GLOBAL VARIABLES - SUPERGLOBALS

```
<html>
<body>

<form method="post" action="<?php echo
$_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>
```

```php
<?php
if ($_SERVER["REQUEST_METHOD"] ==
"POST") {
  // collect value of input field
  $name = $_REQUEST['fname'];
  if (empty($name)) {
    echo "Name is empty";
  } else {
    echo $name;
  }
}
?>

</body>
</html>
```

PHP $_POST

- PHP $_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

PHP $_POST

```
<html>
<body>

<form method="post" action="<?php
echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text"
name="fname">
  <input type="submit">
</form>
```

PHP $_POST

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  // collect value of input field
  $name = $_POST['fname'];
  if (empty($name)) {
    echo "Name is empty";
  } else {
    echo $name;
  }
}
?>

</body>
</html>
```

AUTOGLOBALS (SUPERGLOBALS)

- Autoglobals are predefined global associative arrays whose elements are referred to with a alphanumeric key instead of index number
  - The *$_SERVER* autoglobal
  - The $_POST autoglobal - contains all POST data.
  - The $_GET autoglobal - if we used the method *GET* then our form information would live in the $_GET autoglobal
  - The $_REQUEST autoglobal
    - if you do not care about the source of your request data. It contains the merged information of GET, POST and COOKIE data
  - *$_COOKIE*
  - *$_FILES*
  - *$_ENV*
  - *$_SESSION*

AUTOGLOBALS (SUPERGLOBALS)

```php
<?php
 echo $_SERVER['PHP_SELF'];
 echo $_SERVER['SERVER_NAME'];
 echo $_SERVER['HTTP_HOST'];
 echo $_SERVER['HTTP_REFERER'];
 echo $_SERVER['HTTP_USER_AGENT'];
 echo $_SERVER['SCRIPT_NAME'];
 ?>
```

WHICH BROWSER THE VISITOR USES?

- Browsers send *the user agent* string as part of the HTTP request
- This information is stored in a variable
  $_SERVER['HTTP_USER_AGENT']
- Printing a variable (an array element)

```
<?php echo $_SERVER['HTTP_USER_AGENT']; ?>
```

A sample output of this script may be:

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

- *$_SERVER* is a special reserved PHP variable that contains all web server information
  - It is known as an autoglobal (or superglobal)

IF STATEMENTS

- If statements are used to compare two values and carry out different actions based on the results of the test.
  - If statements take the form IF, THEN, ELSE.
  - The IF part checks for a condition. If it is true, the then statement is executed. If not, the else statement is executed.

```
if (something == something else) {
        then Statement
} else {
        ELSE Statement
}
```

  - For example:

```
if ($username == "webmaster") {
        echo "Enter your password to log on";
} else {
        echo "You are not a recognised user";
}
```

OTHER COMPARISONS

- Compare two different variables to see if their values match e.g.

```
if ($enteredpass == $password)
```

- Use the standard comparison symbols to compare between two variables

```
if ($age < 13), or

if ($date > $finished)
```

- Multiple tests can be checked in one IF statement. For instance,

```
if ($name == "" || $email == "" || $password == "")
{
        echo "Please fill in all the fields";
}
```

SWITCH STATEMENT

```
$destination = "Tokyo";
   echo "Traveling to $destination<br />";
   switch ($destination){
      case "Las Vegas":
         echo "Bring an extra $500";
         break;
       case "Amsterdam":
         echo "Bring an open mind";
         break;
       case "Egypt":
         echo "Bring 15 bottles of SPF 50 Sunscreen";
         break;
       case "Tokyo":
         echo "Bring lots of money";
         break;
       case "Caribbean Islands":
         echo "Bring a swimsuit";
         break;

       default:
         echo "Bring lots of underwear!";
         break;
   }
```

WHILE LOOPS

- The ***WHILE*** loop executes a piece of code until a certain condition is met.

```
$times = 5;
$x = 0;
while ($x < $times) {
    echo "Hello World";
    ++$x; // same as $x = $x + 1;
}
```

- The code above prints "Hello World" 5 times

ARRAYS

- Arrays are special variables which can hold more than one value, each stored in its own numbered 'space' in the array.
- **Setting Up An Array**
  To set up an array with 5 names in it do:

```
$names[0] = 'John';
$names[1] = 'Paul';
$names[2] = 'Steven';
$names[3] = 'George';
$names[4] = ‘Lisa';
```

  To add a value to an array you must specify the location in the array by putting a number in [ ].

USING ARRAYS AND LOOPS

Print numbered names from the array (previous page)

```
$number = 5;
$x = 0;
while ($x < $number) {
    $namenumber = $x + 1;
    echo "Name $namenumber is
$names[$x]<br>";
    ++$x;
}
```

SETTING UP PHP FORMS

Three of the main pieces of code for creating forms are:

- Display a text input box with Your Name written in it as default. The value section of this code is optional. The information defined by name will be the name of this text box and should be unique.

```
<input type="text" name="thebox" value="Your Name">
```

- Will display a large scrolling text box with the text 'Please write your message here.' as default. Again, the name is defined and should be unique.

```
<textarea name="message">
Please write your message here.
</textarea>
```

- create a submit button for your form.

```
<input type="submit" value="Submit">
```

- All the elements for the form must be enclosed in the **<form>** tags.

```
<form action="process.php" method="post">
Form elements and formatting etc.
</form>
```

- The form's action tells it what script to send its data to (in this case its process.php). This can also be a full URL (e.g. http://www.mysite.com/scripts/private/processors/process.php).
- The method tells the form how to submit its data. **POST** sends the data in a data stream to the script when it is requested. **GET** sends the form data in the form of the url so it would appear after a question mark e.g. http://www.mysite.com/process.php?name=david

GETTING THE INFORMATION FROM THE FORM

- Extract variables sent to script using POST method:

  $variablename=$_POST['variable'];

  takes the variable from the POST (the name of a form field) and assigns it to the variable $variablename.

- Similarly, if you are using the GET method you use:

  $variablename=$_GET['variable'];

- This should be done for each variable you wish to use from your form (or URL).

A FORM WITH USER FEEDBACK

- Make a simple form where the user can enter their e-mail address, their name and their comments.

```
<form action="mail.php" method="post">
Your Name: <input type="text"
name="name"><br>
E-mail: <input type="text" name =
"email"><br>
Comments<br>
<textarea name="comments"></textarea><br>
<input type="submit" value="Submit">
</form>
```

MAIL.PHP SCRIPT

```
<?php

$name=$_POST['name'];
$email=$_POST['email'];
$comments=$_POST['comments'];

?>
```

OPERATORS

- Assignment Operators
- Arithmetic Operators
- Comparison Operators
- String Operators
- Combination Arithmetic & Assignment Operators

ASSIGNMENT OPERATORS

- Set a variable equal to a value or set a variable to another variable's value.
  - Assignment is done with the "=", or equal character.

```
$varX = 4;

$varY = $varX
```

- Both $varX and $varY contain the value 4.

ARITHMETIC OPERATORS

Operator	English	Example
+	Addition	2 + 4
-	Subtraction	6 – 2
*	Multiplication	5 * 3
/	Division	15 / 3
%	Modulus	43 % 10

COMPARISON OPERATORS

Operator	English	Example	Result
==	Equal To	$x == $y	false
!=	Not Equal To	$x != $y	true
<	Less Than	$x < $y	true
>	Greater Than	$x > $y	false
<=	Less Than or Equal To	$x <= $y	true
>=	Greater Than or Equal To	$x >= $y	false

COMBINING ARITHMETIC & ASSIGNMENT OPERATORS

$counter = $counter + 1; can be shortened to $counter += 1;

Operator	English	Example	Equivalent Operation
+=	Plus Equals	$x += 2;	$x = $x + 2;
- =	Minus Equals	$x -= 4;	$x = $x - 4;
*=	Multiply Equals	$x *= 3;	$x = $x * 3;
/=	Divide Equals	$x /= 2;	$x = $x / 2;
%=	Modulo Equals	$x %= 5;	$x = $x % 5;
.=	Concatenate Equals	$my_str .= "hello";	$my_str = $my_str . "hello";

STRING OPERATORS

The period "."

- is used to add two strings together
- is the concatenation operator for strings.

Example PHP Code:

```
$Fname = "Bill ";
$SName = "Gates!";
$FullName = $FName ." " . $SName;
echo $FullName . "!"; // displays Bill Gates!
```

Display:

Bill Gates!

PRE/POST-INCREMENT & PRE/POST-DECREMENT

- Pre/Post-Increment: to add one to a variable or "increment" use the "++" operator:

  $x++; is equivalent to

  $x += 1; or

  $x = $x + 1;

- Pre/Post-Decrement: to subtract 1 from a variable, or "decrement" use the "--" operator:

  $x--; is equivalent to

  $x -= 1; or

  $x = $x - 1;

- You can specify whether you want the increment to happen before the line of code is being executed or after the line has been executed.

THE INCLUDE FUNCTION

- Takes a file name and simply inserts that file's contents into the script that calls the *include* function.
  - Type up a common ***header*** or ***menu*** file that you want all your web pages to include. When you add a new page to your site, instead of having to update the links on several web pages, you can simply change the Menu file.
- Included files usually have "***.inc***" extension.

MENU.INC

menu.inc // Suppose "***menu.inc***" is a common menu file

```
<html> <body>
<a href="index.php">Home</a> -
<a href="about.php">About Us</a> -
<a href="links.php">Links</a> -
<a href="contact.php">Contact Us</a> <br />
```

index.php

```
<?php include("menu.inc"); ?>
<p>A home page that uses a common menu to save time when
adding new pages!</p>
</body>
</html>
```

display

Home - About Us - Links - Contact Us

A home page that uses a common menu to save time when adding new pages!

WHAT A VISITOR SEES?

The include function is the same as copying and pasting, viewing the source of *index.php,* the visitors would see:

index.php

```
<html> <body>
<a href="index.php">Home</a> -
<a href="about.php">About Us</a> -
<a href="links.php">Links</a> -
<a href="contact.php">Contact Us</a> <br />


 <p>A home page that uses a common menu to save time when adding new pages!</p>

</body>
</html>
```

PHP REQUIRE FUNCTION

- The require function is also used to include a file into your PHP code.
- When you include a file with the *include* function and PHP cannot find it, an error message is displayed, but the rest of the script is executed.
  - **::Warning does not prevent our PHP script from running.::**

- When you include a file with the *require* function and PHP cannot find it an error message is displayed and the rest of the script is NOT executed, i.e. the script dies after the *require* function returns an error!
- Therefore, *require* function is preferred to *include* function because your scripts should not be executing if necessary files are missing or misnamed.

REQUIRE VS. INCLUDE

- Using ***include***

```
<?php
        include("noFile.inc");
        echo " The rest of the script!";
?>
```

Display: ***Warning*** *message*
The rest of the script!

- Using ***require***

```
<?php
        require("noFile.php");
        echo "Hello World!";
?>
```

Display: ***Warning*** *message*

FUNCTION DECLARATION

```
<?php
        function displayString() {
                echo “Function displays a string:<br />";
        }
function displayString1($str=“Hello”) {
                echo “Function displays a string:<br />“. $str;
        }

        displayString(); // invoke the function
         displayString1(“Hello World”);
?>
```

FUNCTION PARAMETERS

Parameters – information that functions can use, provided from outside the function

- Parameter is the string to be displayed

```
<?php
        function displayString( $str ) {
                echo "Function displays a string:" . $str . "<br />";
        }

        displayString("Hello Word"); // invoke the function
?>
```

RETURN VALUES FROM FUNCTIONS

A function usually returns a value. A function can only return one thing/value (integer, float, array, string, etc.)

```
<?php
        function sum($X, $Y) {
                $Z = $X + $Y;
                return $Z;
        }
function sumR(&$X, &$Y) {
                $Z = $X + $Y;
                return $Z;
        }

        $myNumber = 0;
        echo "Initially myNumber = ". $myNumber ."<br />"; // 0

         $myNumber = sum(3, 4);
        echo "3 + 4 = " . $myNumber ."<br />"; // 7
?>
```

PHP BUILT-IN FUNCTIONS

Other built-in functions such as

- **strpos**
- **strtoupper()**
- **strlen()**
- …and many more (http://www.w3schools.com/php/php_ref_string.asp)

POST

```
<form action="mail.php" method="post">

Name: <input type="text" name="name"><br>
E-mail: <input type="text" name = "email"><br><br>
<textarea name="“ebody"></textarea><br><br>
<input type="submit" value="Submit">
</form>
```

- A simple form where the user enters e-mail address, name and email body.
- The form data is submitted to the “mail.php" using the POST method.
  - PHP stores all the "posted" values into an *associative array* called "$_POST".
  - The names of the form data names represent the *keys* in the "$_POST"

MAIL.PHP SCRIPT

```
<?php
        $name = $_POST['name'];
        $email = $_POST['email'];
        $ebody = $_POST[‘ebody'];

        }
?>
```

GET

```
<form action="mail.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name = "email"><br><br>
<textarea name="ebody"></textarea><br><br>
<input type="submit" value="Submit">
</form>
```

- The form data is submitted to the "mail.php" using the GET method.
  - The *get* method passes the variables along to the "mail.php" web page by appending them onto the end of the URL. The URL, after clicking submit, would have this added on to the end of it:
  - "?name=##&email=##&ebody=##"
  - The "?" tells the browser that the following items are variables. The "mail.php" code must also use the "$_GET" associative array.
- The *get* method displays the variable information to your visitor
  - Do not send password information or other sensitive items with the *get* method.

PHP FILES MANIPULATION

- Creating
- Opening
- Closing

- Reading
- Writing
- Appending
- Truncating
- Deleting

- Uploading and Downloading

FILE CREATION AND OPENING

The ***fopen*** function is used to *open* or *create* files.

Parameters: file name, access mode

- **Open:** if a file specified exists it is opened
- **Create:** if a file is not **found** (it does not exist) it will be created, given that you open the file for writing or appending.

Example:

```
$fileName = "myFile.txt";
$fileHandle = fopen($fileName, 'w');
fclose($fileHandle);
```

FILE OPENING MODES

- PHP requires user to specify intentions when opening file.
  - **Read: 'r'** Open for read only. The file pointer begins at the front (start) of the file.
  - **Write: 'w'** Open for write only. The data in the file is erased and writing begins at the **beginning** of the file. This is also called truncating a file. The file pointer begins at the start of the file.
    - **write a new file, or overwrite an existing file**
  - **Append: 'a'** Open for write only. The data in the file is preserved and writing begins at the end of the file. The file pointer begins at the end of the file.
- Reading or writing functions begin at the location specified by the file pointer.

FILE OPENING MODES (CTD.)

- Opening a file such that reading and writing is allowable!
  - Place a plus sign "+" after the file mode character.
- **Read/Write: 'r+'**
  - Opens a file so that it can be read from and written to.
  - The file pointer is at the beginning of the file.
- **Write/Read: 'w+'**
  - the same as r+, **except** that it deletes all information in the file when the file is opened.
- **Append: 'a+'**
  - the same as r+, **except** that the file pointer is at the end of the file.

CLOSE

- The function *fclose* closes an opened file
  - requires the file handle that we want to close down.
- After a file has been closed down with *fclose* it is impossible to
  - read,
  - write or
  - append to that file

  unless it is once more opened up with the *fopen* function.

WRITE

- The *fwrite* function allows data to be written to any type of file. It takes these parameter
  - 1st is the file handle
  - 2nd is the string of data that is to be written.
- Below we are writing a couple of names into our test file *testFile.txt* and separating them with a carriage return.

Example:

```
$fileName = "myFile.txt";
$fileHandle = fopen($fileName, 'w') or die("can't open file");


$msg = "Students are hardworking\n";
fwrite($fileHandle, $msg);

$msg = "But Computer Science is challenging … \n";
fwrite($fileHandle, $msg);


fclose($fileHandle);
```

READ

- The *fread* function is used to read from files Parameters are:
  - a file handle
  - an integral number of bytes to read.

Example:

```
$fileName = "myFile.txt";
$fileHandle = fopen($fileName, 'w') or die("can't open file");
$msg = "Students are hardworking\n";
fwrite($fileHandle, $msg);
fclose($fileHandle);

$fileHandle = fopen($fileName, 'r');
$msg = fread($fileHandle, 24);
fclose($fileHandle);
echo $msg;
```

READ: *GETS* FUNCTION

gets read a line of data at a time from a file

- If data is separated with new lines then you could read in one segment of data at a time with the *gets* function.
- The *fgets* function searches for the first occurrence of "\n", the newline character

Example:

```
$fileName = "myFile.txt";
$fileHandle = fopen($fileName, 'r');
$msg = fgets($fileHandle);
fclose($fileHandle);
echo $msg;
```

READ

This code reads the whole file

Example:

```
$fileName = “myFile.txt";
$fileHandle = fopen($fileName, ‘r');
$msg = fread($fileHandle, fileSize($fileName));
fclose($fileHandle);
echo $msg;
```

DELETE FILES

A file is deleted by calling the *unlink* function.

- unlinking a file makes the system forget about it or delete it!
- Close the file with a *fclose* function before deleting it

Example:

```
$fileName = "myFile.txt";
$fileHandle = fopen($fileName, 'r');
$msg = fread($fileHandle, fileSize($fileName));
fclose($fileHandle);
echo $msg;
unlink($fileName);
```

APPEND

- To add on to a file, open it up in append mode.

Example

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'a') or die("can't open file");

$stringData = "New Stuff 1\n";
fwrite($fh, $stringData);

$stringData = "New Stuff 2\n";
fwrite($fh, $stringData);
fclose($fh);
```

TRUNCATING

```
$myFile = "testFile.txt";
$fh = fopen($myfile, 'w');
fclose($fh);
```

- The code above erase the content of the file “testFile.txt”. Opening a file for writing with the paramater 'w' wipes all data from that file. This action is also referred to as "truncating" a file.

FILE UPLOAD

A form for selecting a file to be uploaded

On the client side: upload.html

```
<form enctype="multipart/form-data" action="uploader.php“
method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
Choose a file to upload: <input name="uploadedfile" type="file"
/><br />
<input type="submit" value="Upload File" />
</form>
```

- The enctype attribute specifies how the form-data should be encoded when submitting it to the server.
- **Note:** The enctype attribute can be used only if method="post".

FILE UPLOAD (CTD.)

On the server side: *uploader.php*

- The uploaded file exists in a temporary storage area on the server. *tmp_name* contains the path to the temporary file that resides on the server.
- To save the file we use the *$_FILES* associative array (PHP stores all information about files in *it)*
- *uploadedfile* is the name of the file to upload as seen in HTML form. *name* contains the original path of the user uploaded file.

$_FILES['uploadedfile']['name']
$_FILES['uploadedfile']['tmp_name']

On the server side: uploader.php

```
// Where the file is going to be placed
// $target_path = "uploads/";

/* Add the original filename to our target path. Result is
"uploads/filename.extension" */
// $target_path = $target_path . basename(
$_FILES['uploadedfile']['name']);

/* This is how we will get the temporary file...*/
// $_FILES['uploadedfile']['tmp_name'];


$target_path = "uploads/";


$target_path = $target_path . basename( $_FILES['uploadedfile']['name']);
if( ($_FILES['uploadedfile']['tmp_name'], $target_path)) {
    echo "The file ". bamove_uploaded_filesename(
$_FILES['uploadedfile']['name']). " has been uploaded";
} else{

    echo "There was an error uploading the file, please try again!";
}
```