PHP Hypertext preprocessor

COURSE MATERIALS

You can access the course materials via this link

http://goo.gl/ev41na

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";
```