CSC 337

Introduction to PHP

Rick Mercer

URLs and web servers

http://server/path/file

- Usually when you type a URL in your browser:
  - your computer looks up the server's IP address using a Domain Name Server (DNS)
  - Your browser connects to that IP address and requests the given file
  - Web server software (e.g. Apache) locates that file from the server's local file system, and sends back its contents back to your browser

Server-Side web programming

- Server-side pages are programs written using one of many web programming languages/frameworks
  - examples: PHP, Java/JSP, Ruby on Rails, Node.js ASP.NET, Python, Perl
- The web server contains software that allows it to run programs on the server and send back their output
  - Often accessing files and databases
- Each language/framework has its pros and cons

Why PHP?

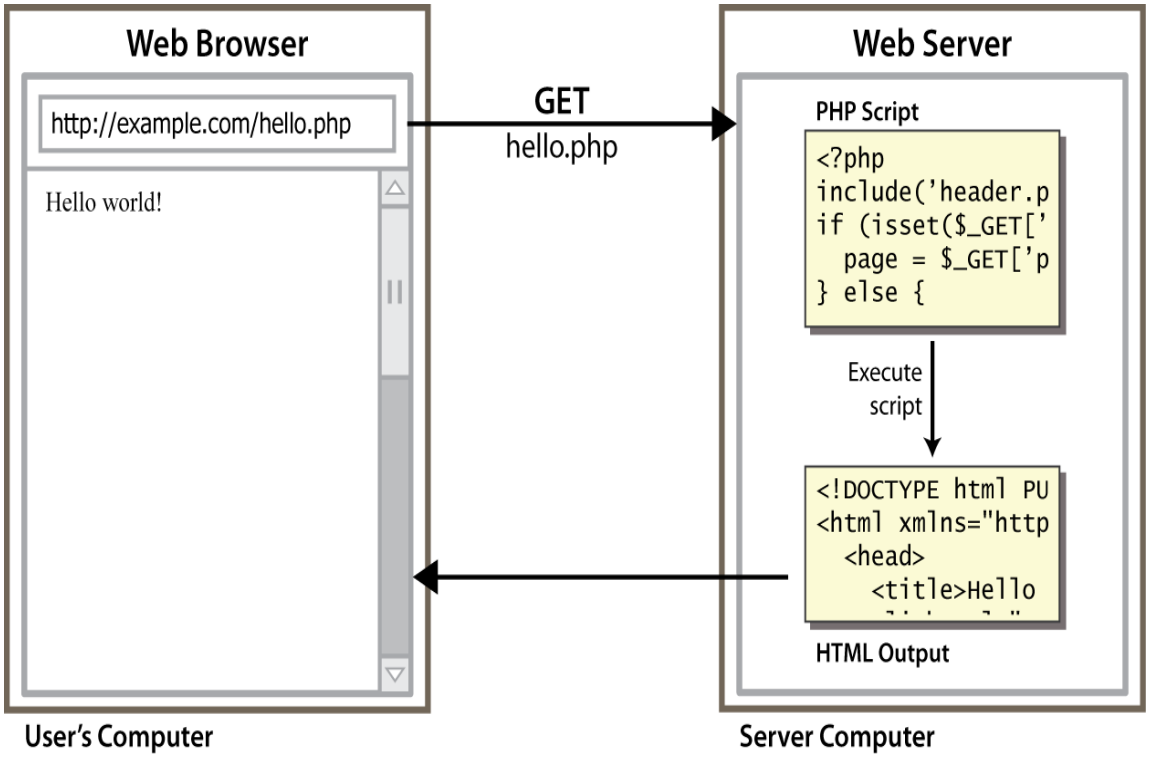
- There are many other options for server-side languages (Ruby on Rails, JSP, ASP.NET, …) Why choose PHP?
  - Free and open source: anyone can run a PHP-enabled server free of charge
  - **compatible:** supported by most popular web servers
  - **simple:** lots of built-in functionality; familiar syntax
  - **available:** installed on CS servers, most commercial web hosts,
  - **Easy to install:** XAMPP gets you a local server which makes everything easier
  - **well-documented:** type php.net/*functionName* in browser Address bar to get docs for any function

What is PHP?

- **PHP** stands for "PHP Hypertext Preprocessor"

- It's a server-side scripting language
- Used to make web pages dynamic:
  - provide different content depending on context
  - interface with other services: database, e-mail,
  - authenticate users
  - process form information
    - You probably fill out several forms a week

Lifecycle of a PHP web request



- Browser requests a .html file (**static content**): server just sends that file
- Browser requests a .php file (**dynamic content**): server reads it, runs code inside it, sends back html page to your browser

Let us now learn PHP with no server or HTML
Our 4th language

- Contents could go into a file hello.php:
- A block or file of PHP code begins with <?php and ends with ?>
- PHP statements, function declarations, etc. appear between these endpoints

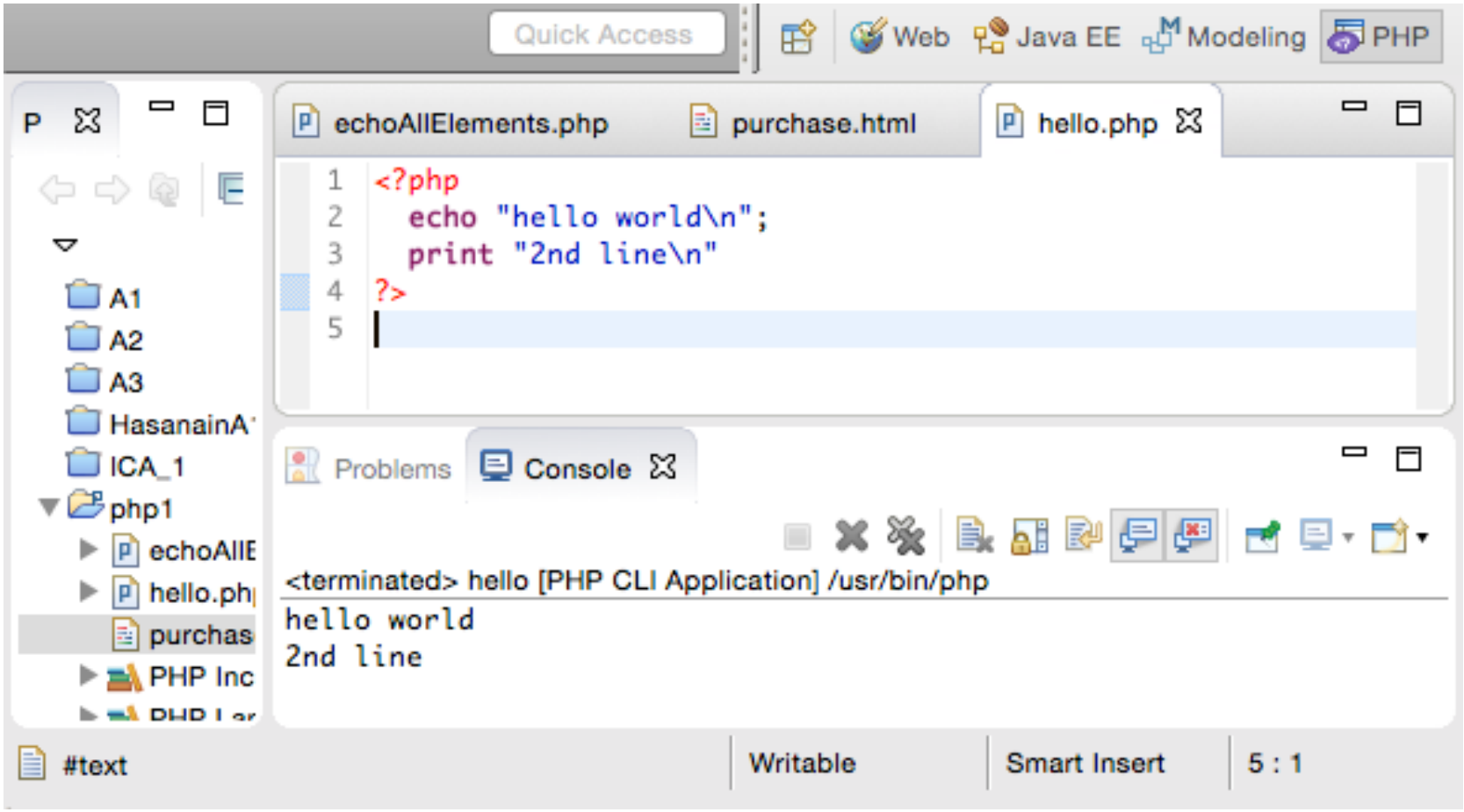
```
<?php
  echo "hello world\n";
  print "2nd line" . PHP_EOL;
  echo "third";
?>
```

Output:

```
hello world
2nd line
third
```

Viewing PHP output

- Install XAMPP
- In Eclipse select **File > New > Other > PHP > PHP Project**
- Select **Run > Run As > CLI** *Command Line Interface*



Print or echo

```
<?php
  print "Hello, World!\n";
  echo "Escape \"chars\" are the SAME as in Java!\n";
  print "You can have line breaks in
           a string.";
  echo 'A string can use "double or single-quotes"';
?>
```

Output:

```
Hello, World!
Escape "chars" are the SAME as in Java!
You can have line breaks in
      a string.A string can use "double or single-quotes"
```

Viewing PHP output

- The ability to use ' and " will help when we start echoing html back to the browser

```
<?php
echo '<div class="float-left">'. PHP_EOL . '   <img
src="images/critic.gif" alt="Critic" />'. PHP_EOL .'</div>'
?>
```

Output:

```
<div class="float-left">
   <img src="images/critic.gif"  alt="Critic" />
</div>
```

A few PHP details

- Start must start with $
  - Other rules are like Java (e.g. case sensitive)
- Concatenation with . rather than +
- Arithmetic operators

```
+   -   *   /   %   ++   --   =   +=   -=   *=   /=   %=
```

- Loosely typed: no need to declare type (int, String)

```
<?php
$x = 8;
$y = 7;
echo  $x . " + " . $y . " = ". ($x + $y);
?>
```

Output:

```
8 + 7 = 15
```

Output when using '+' instead of '.' is 30 (ouch)

```
echo $x + " + " + $y + " = " + ($x + $y);
```

Types

```
                                    Output to console
echo gettype ( 1 );                                 integer
echo gettype ( 1.2 );                               double
echo gettype ( TRUE );                              boolean
echo gettype ( FALSE );                             boolean
echo gettype ( 'true' . 'false' );                  string
echo gettype ( 'true' + 'false' );                  integer
echo gettype ( [1, 2, 3, 4]  );                     array
```

Arithmetic Operators

```
echo 1 + 2;         3
echo 5 / 2;       2.5  No integer division
echo 5 % 2;         1   Remainder
echo 4 * 1.2;     4.8
echo 4 - 5.2;    -1.2
echo (0 / 2);       0
echo (TRUE || FALSE);    1 for TRUE
echo (TRUE && FALSE);    No OUTPUT!!!!!!!!
```

A Few Built-in Functions

```
/*
 * Multi-line
 * comment
 */
echo abs(5-7);    2

echo round(2.5);  3

echo sqrt(16);    4

// Assume the current date is 19-June
echo date ( "Y/m/d" ) . "\n";    2017/06/19

# Another one line comment with #
echo date ( "d/M/y" ) . "\n";    19/Sep/17
```

If statements

```
$num = 124;
if($num < 0 || $num > 100)
   print("Out of range");
else
   print("In range");
```

```
$grade = 72;
if ($grade >= 90)
    print ("A") ;
else if ($grade >= 80)
    print ("B") ;
else if ($grade >= 70)
    print ("C") ;
else if ($grade >= 60)
    print ("D") ;
```

Loops

```php
<?php
for($count = 0; $count <= 3; $count ++) {
  print 'pow(' . $count .', 2) = ' . pow ( $count, 2 ) . PHP_EOL;
}
?>
```

```
pow(0, 2) = 0
pow(1, 2) = 1
pow(2, 2) = 4
pow(3, 2) = 9
```

```php
<?php
$str = 'Arizona';
$i = 0;
while ( $str[$i] != 'z' ) {
  print substr($str, $i) . PHP_EOL;
  $i += 1;
}
?>
```

```
Arizona
rizona
izona
```

The assert function

- The assert function wants a Boolean expression
- One of two things happen
  - Nothing when the expression is true

```
assert(4.0 == sqrt(16));
assert(3 == round(2.5));
assert(2 == round(2.499999));
assert(2 == abs(5-7));
```

  - See a message when false

```
assert(3.9999 == sqrt(16));
```

```
PHP Warning:  assert(): Assertion failed in
/Applications/XAMPP/xamppfiles/htdocs/Three/10Functions.php on line 66
```

String Functions via asserts

```
// All of these assert methods pass
assert ( 65 == ord ( 'A' ) );
assert ( 'A' == chr ( 65 ) );

$str = 'Arizona';
assert ( 7 == strlen ( $str ) );              // length of a
string
assert ( 'ARIZONA' == strToUpper ( $str ) );
assert ( 'zona' == substr ( $str, 3 ) );     // part of a string
assert ( 'na' == substr ( $str, 5 ) );
assert ( 4, strpos ( $str, 'ona' ) );
assert ( 'anozirA' == strrev ( $str ) );

assert ( 'abcdef'[2] == 'c' ); // One character
assert('abcdef'[99] == '');    // No error, just an empty string
```

PHP Functions

```
function name(parameterName, ..., parameterName) {
  statements;
}
```

```
<?php
echo 'BMI at 160 pounds, 70 inches: ' . bmi(160, 70) . PHP_EOL;

function bmi($weight, $height) {
    $result = 703 * $weight / $height / $height;
    return $result;
}

echo 'BMI at 160 pounds, 68 inches: ' . bmi(160, 68);
?>
```

```
BMI at 160 pounds, 70 inches: 22.955102040816
BMI at 160 pounds, 68 inches: 24.325259515571
```

- Parameter types and return types are not needed
- A function with no return statements is "void"
- Declared in any PHP block, start/end/middle

PHP Value Parameters

- PHP has value parameters
  - Pass by value, the default
  - Can not change the argument

```
function noChange($str) {
  $str = $str . ' extra';
}

$str = 'original';
assert('original' == $str);
assert(8 == strlen($str));

noChange($str);

assert('original' == $str);
assert(8 == strlen($str));
```

PHP Reference Parameters

- PHP also has reference parameters
  - Pass by reference? Add **&**
  - Changes the argument
  - 337 Probably won't need them

```
function change(& $str) {
  $str = $str . ' extra';
}

$str = 'original';
assert( 8 == strlen($str) );

change( $str );

assert( 'original extra' == $str );
assert( 14 == strlen($str) );
```

Default arguments

- PHP has default arguments
  - 337 probably won't need them

```
function makecoffee($type = "cappuccino") {
  // PHP variables inside a string evaluate to the value
  // Notice $type appears as three different values, not '$type'
  return "Making a cup of $type.\n";
}

echo makecoffee();
echo makecoffee(null);
echo makecoffee("espresso");
```

Output:

Making a cup of cappuccino.

Making a cup of .

Making a cup of espresso.

PHP Arrays

- PHP has three types of arrays
  - Indexed, Associative, Multidimensional
- Indexed Array: Arrays with sequential numeric index, such as 0,1,2 etc.

```
$arr = array (
    'Kim',
    'Chris',
    'Dakota'
);

assert ( 'Kim' == $arr [0] );
assert ( 'Chris' == $arr [1] );
assert ( 'Dakota' == $arr [2] );
```

Associative Arrays

- Associative arrays are arrays that use named keys mapped to values

```php
<?php
  // This is a complete PHP program
  $array = array (
     "key1" => "value1",
     "two" => "another",
     "three" => "we will use associative arrays"
  );

  assert( $array["key1"] == "value1" );
  assert( $array["key1"] != "key1" ) ;
  assert( $array["two"] == "another" );
  assert( $array["three"] == "we will use associative arrays" );
?>
```

For loops on indexed arrays

- Subscripted with [ ]
- Don't forget the $

```
$array = array (5, 6, 1, 0, 2);
$sum = 0;
for($i = 0; $i < count($array); $i++) {
  $sum += $array[$i];
}
assert( $sum == 14 );
```

foreach loops on Associative Arrays

- Use example below
- This $array is initialized in a different way

```php
<?php
$array["key1"] = "value1";
$array["two"] = "another";
$array["three"] = "third";

$values = "";
foreach($array as $val) {
   // Use '.' to concatenate. NOT '+'
   $values = $values . $val . '_';
}
assert ( $values == 'value1_another_third_' );
?>
```

array_push

- Can add elements at the end of the array

```
$nums[0] = 99;
array_push($nums, 88);
array_push($nums, 77, 66, 55);
foreach($nums as $element) {
    echo $element . PHP_EOL;
}
```

Output:

```
99
88
77
66
55
```

Splitting/joining strings

- Use `explode` and `implode` to convert between strings and arrays

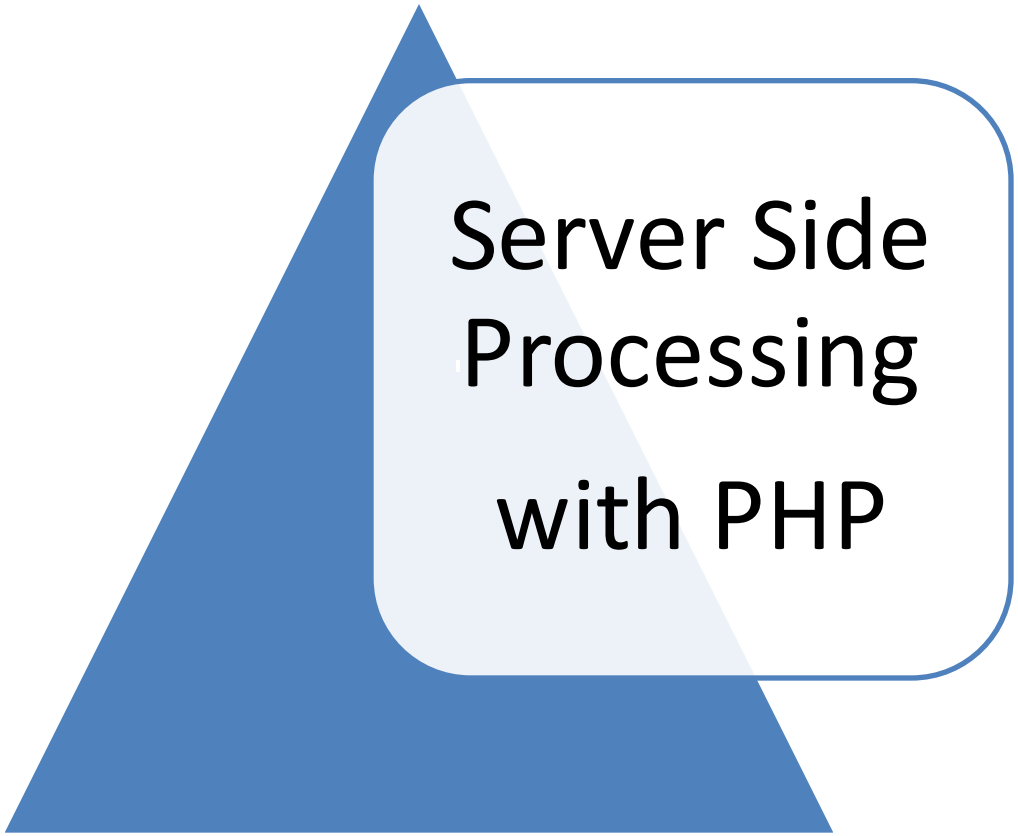
```
$array = explode(delimiter, string);
$string = implode(delimiter, array);
```

```php
<?php
$s = "CSC 337 UofA";
$a = explode ( " ", $s );
echo $a [0] . PHP_EOL;
echo $a [1] . PHP_EOL;
echo $a [2] . PHP_EOL . PHP_EOL;

$arr = array(1, 2, 3, 4, 5, 6, 7);
$str = implode("|", $arr);
print $str;
?>
```

output

```
CSC
337
UofA

1|2|3|4|5|6|7
```

CSC 337



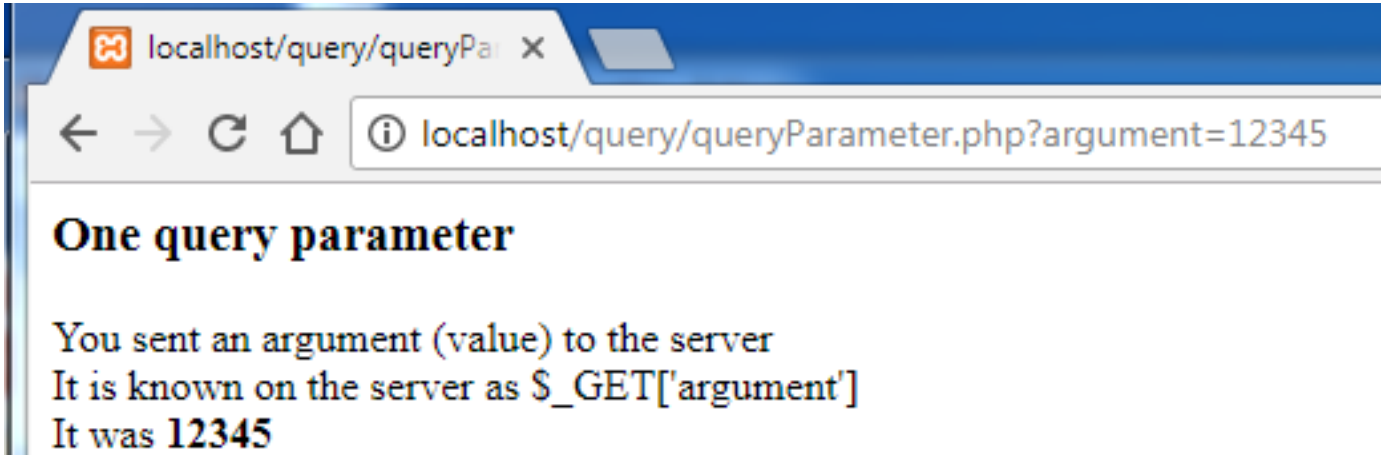
Query Parameters

- We can pass values from the browser to a PHP page on the server
  - Query parameters are at the end of the url
  - General form: `url?parameter=value`
- On the server, use global array $_GET
- Value of the argument passed from the browser to the server is stored in `$_GET['parameter'];`

The PHP file

- We can pass values from the browser to a PHP page on the server
  - Query parameters are at the end of the url
  - General form: `url?parameter=value`

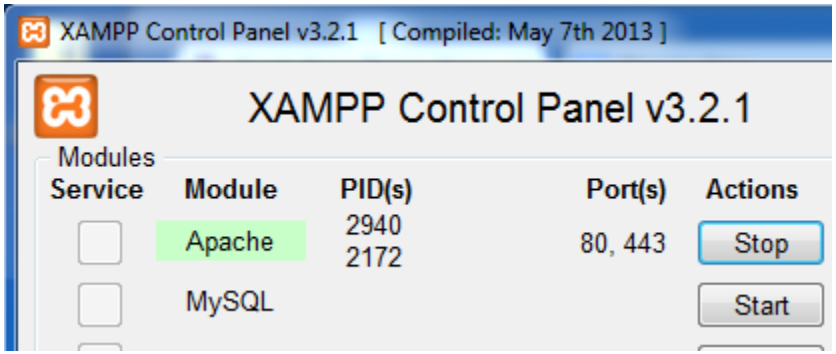
```php
<?php
echo "<h3>One query parameter</h3>";
echo "You sent an argument (value) to the server<br>";
echo "It is known on the server as \$_GET['argument']<br>";
echo "It was <b>" . $_GET['argument'] . "</b>";
?>
```



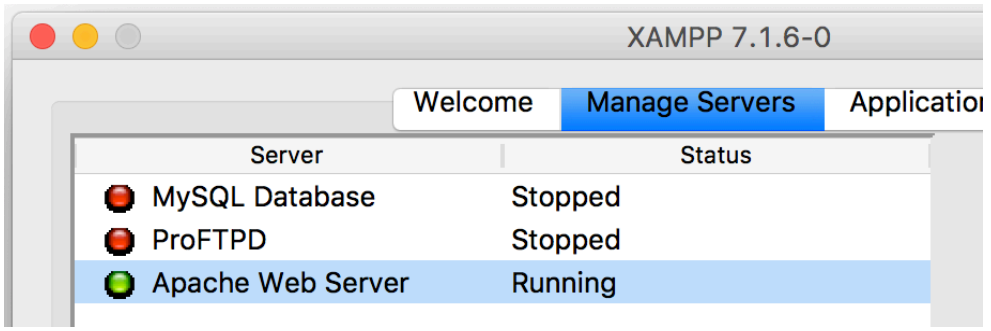
Need the Server Running

- Start XAMMP and the Apache Server

Windows



Mac



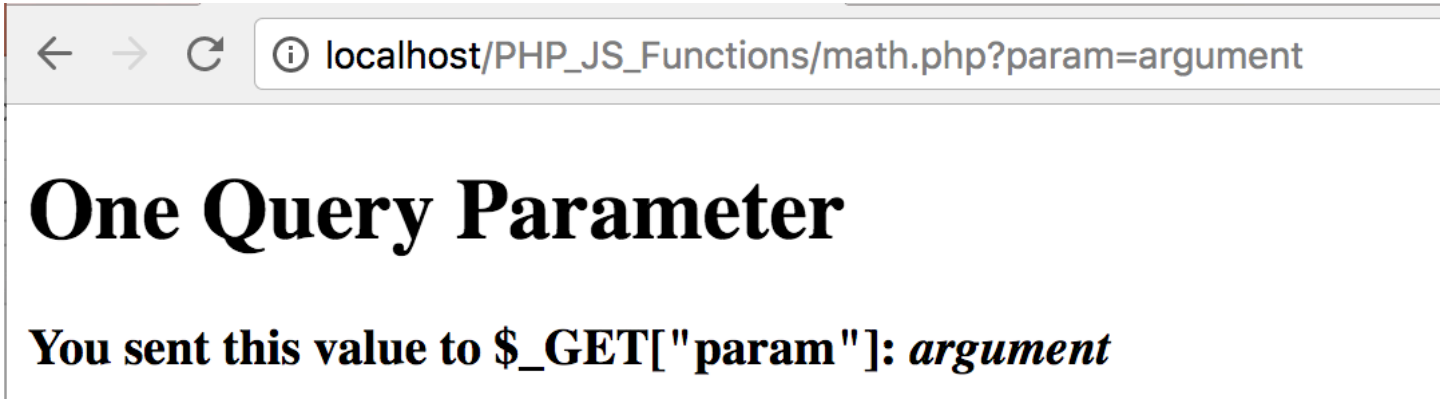
- Load this url in Chrome, change folder name 'query' to you project name

```
http://localhost/query/queryParameter.php?argument=12345
```

Can mix PHP and HTML in PHP files

- We need to start the server, run XAMPP

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Query Parameters</title>
</head>
<body>
<h1>One Query Parameter</h1>
<?php
$value = $_GET ["param"];
echo '<h3>You sent this value to $_GET["param"]: <i>'  . $value
. '</i></h3>';
?>
</body>
</html>
```



Code Demo

- Write a PHP file that writes html back so the page looks like this with the two given query parameters

11 + 7 = 18
11 - 7 = 4
11 / 7 = 1.57142857142857
11 * 7 = 77
11 % 7 = 4

Code Demo

- Start with this code in math.html

```
<body>

<h3> Query parameters sent to PHP</h3>

<form action="math.php" method="get">
a: <input type="text" name="a" required> <br>
b: <input type="text" name="b"  required> <br>
<input type="submit">
</form>

</body>
```

- Start with this code in math.php

```
<!DOCTYPE html>
<html>
<head>
<title>Mixing PHP and HTML</title>
</head>
<?php
  $a = $_GET['a'];
  $b = $_GET['b'];
?>
```

- Complete with a mix of HTML and PHP using a new shorter PHP

  block <?= echo 'Hello' ?>

```
<!DOCTYPE html>
<html>
<head>
<title>Mixing PHP and HTML</title>
</head>
<?php
  $a = $_GET['a'];
  $b = $_GET['b'];
?>
```

localhost/folder/math.php?a=17&b=9

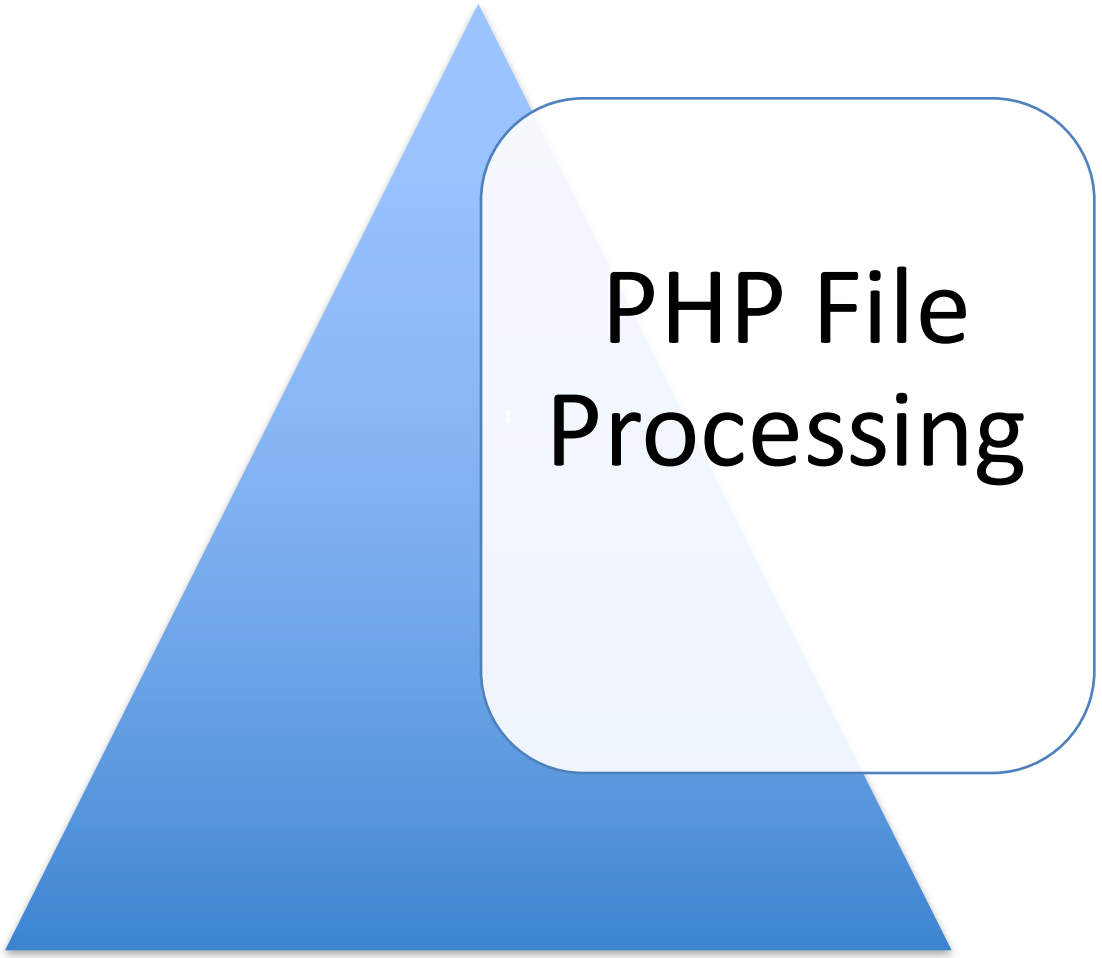
```
<!DOCTYPE html>  <!- file name "math.php"
<html>
<head>
<title>Mixing PHP and HTML</title>
</head>
<?php
  $a = $_GET['a'];
  $b = $_GET['b'];
?>
<body>
 <h4>Some math</h4> <!--  New short PHP block ->
 <p> <?= $a ?> + <?= $b ?> = <?=($a + $b) ?> </p>
 <p> <?= $a ?> - <?= $b ?> = <?=($a - $b) ?> </p>
<?php // Ugly mix above; echo with HTML in strings below is cleaner
   echo $a . ' *  ' . $b . ' = ' . ($a * $b) . "<br>";
   echo $a . ' /  ' . $b . ' = ' . ($a / $b) . "<br>";
   echo $a . ' %  ' . $b . ' = ' . ($a % $b) . "<br>";
?>
</body>
</html>
```

Some math

17 + 9 = 26

17 - 9 = 8

17 * 9 = 153
17 / 9 = 1.8888888888889
17 % 9 = 8


PHP File
Processing

Reading files line by line

- Use `fopen` to create a new file
- Use `fgets` to read an entire line
  - Use this read as the boolean expression that will be false when the file has no more lines
- `fclose` the file

```
$file = fopen ( "small.txt", "r" );
while ( ($line = fgets ( $file )) ) {
  echo $line;  // each $line contains an end of line
}
fclose ( $file );
```

Input file and the output, which match

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.
```

Writing files line by line

- Use `fopen` to create a new file
- Use `fprintf` of `fwrite` to write to that new file
  - The one below formats integers to 3 spaces
- `fclose` the file

```
$file = fopen ( "table", 'w' );
for($row = 1; $row <= 5; $row ++) {
  for($col = 1; $col <= 12; $col ++) {
      fprintf($file, "%3s",  $row * $col);
  }
  fwrite( $file, PHP_EOL );
}
fclose ( $file );
```

output in a file

```
  1   2   3   4   5   6   7   8   9  10  11  12
  2   4   6   8  10  12  14  16  18  20  22  24
  3   6   9  12  15  18  21  24  27  30  33  36
  4   8  12  16  20  24  28  32  36  40  44  48
  5  10  15  20  25  30  35  40  45  50  55  60
```

Other file I/O functions

function names	**category**
file file_get_contents file_put_contents	reading/writing entire files
basename, file_exists, filesize, fileperms, filemtime, is_dir, is_readable, is_writable, disk_free_space	asking for information
copy, rename, unlink, chmod, chgrp, chown, mkdir, rmdir	manipulating files and directories
glob scandir	reading directories to get an array of file names

Read each line as an array element

- `file("filename")` returns lines of the file as one array of string elements
  - \n remains at end of each string!!!
    - PHP_EOL not needed

```
<?php
$arr = file("input.txt");
for($line = 0; $line < count($arr); $line++)
   print($line . ': ' . $arr[$line]);
?>
```

Input file input.txt

```
Hello
how r u?
Can you tell
me your name
```

Output

```
0: Hello
1: how r u?
2: Can you tell
3: me your name
```

Read an entire file as one string

- `file_get_contents("filename")` returns the entire contents of a file as a single string
  - has newlines from the input file

```
<?php
   $str = file_get_contents ( "bar.txt" );
   print ($str);
 ?>
```

Input file bar.txt

```
One two three
four five
six
```

Output

```
One two three
four five
six
```

Unpacking an array: `list`

- The list function "unpacks" an array into a set of variables you declare
- When you know a file or line's exact length/format, use file and list to unpack it

```
list($var1, ..., $varN) = array;
```

```
<?php
list( $name, $year, $rating ) = file ( "info.txt" );
echo $name;
echo $rating . '%' . "\n";
echo 'Released in ' . $year;
?>
```

Input file info.txt

```
Bourne Identity
2002
93
```

Output

```
Bourne Identity
93%
Released in 2002
```

Write a big string to a file

- `file_put_contents("filename", $content)` write the entire string `$content` to `"filename"`
  - has newlines from the input file

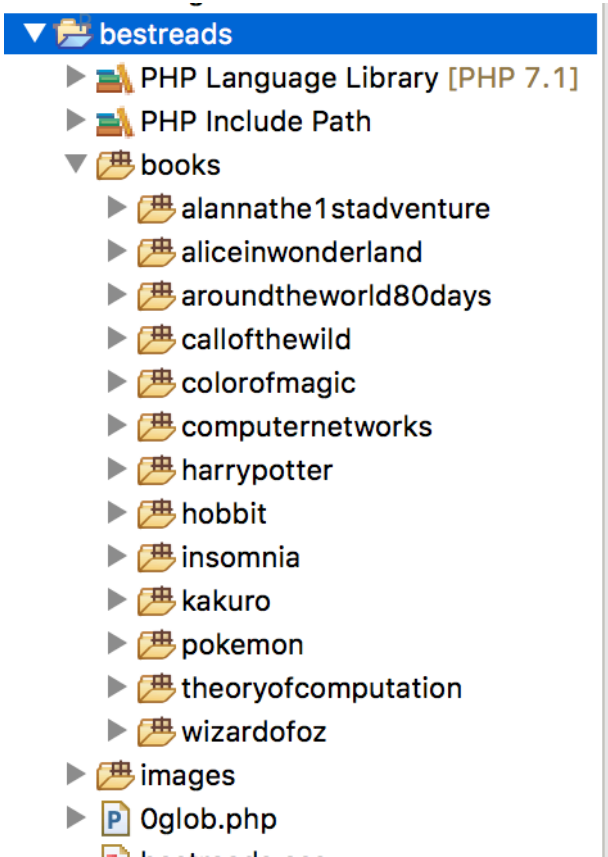
```
file_put_contents ("fname", "a\nBB\nccc\nDDDD" );
$str = file_get_contents ( "fname" );
print_r($str);
```

Output

```
a
BB
ccc
DDDDD
```

glob: to get an array of file names

```php
<?php
$arrayOfFileNames = glob('books/*');
print_r($arrayOfFileNames);
?>
```




```
Array
(
    [0] => books/alannathe1stadventure
    [1] => books/aliceinwonderland
    [2] => books/aroundtheworld80days
    [3] => books/callofthewild
    [4] => books/colorofmagic
    [5] => books/computernetworks
    [6] => books/harrypotter
    [7] => books/hobbit
    [8] => books/insomnia
    [9] => books/kakuro
    [10] => books/pokemon
    [11] => books/theoryofcomputation
    [12] => books/wizardofoz
)
```

Code Demo

- Write a mix of PHP and HTML to generate HTML to show the list of all files where the PHP is located

1. .
2. ..
3. .DS_Store
4. .buildpath
5. .project
6. .settings
7. NewFile.html
8. functions.php
9. images
10. mortalkombat
11. movies.css
12. princessbride
13. review.php
14. test.php
15. testFunctions.php
16. tmnt
17. tmnt.html
18. tmnt2

Working . . .

Answer: run as PHP CLI

```php
$array = localFiles();
echo '<ol>';
for($i = 0; $i <sizeOf($array); $i++) {
   echo '<li>' . $array[$i] . '</li>' . PHP_EOL;
}
echo '</ol>';
?>
```

1. .
2. ..
3. .DS_Store
4. .buildpath
5. .project
6. .settings
7. NewFile.html
8. functions.php
9. images
10. mortalkombat
11. movies.css
12. princessbride
13. review.php
14. test.php
15. testFunctions.php
16. tmnt
17. tmnt.html
18. tmnt2

```
<ol><li>.</li>
<li>..</li>
<li>.DS_Store</li>
<li>.buildpath</li>
<li>.project</li>
<li>.settings</li>
<li>NewFile.html</li>
<li>functions.php</li>
<li>images</li>
<li>mortalkombat</li>
<li>movies.css</li>
<li>princessbride</li>
<li>review.php</li>
<li>test.php</li>
<li>testFunctions.php</li>
<li>tmnt</li>
<li>tmnt.html</li>
<li>tmnt2</li>
</ol>
```