More PHP

PHP Version Differences

- PHP 5.0 requires use of the _REQUEST or _GET or _POST variables to access variables passed in by forms
- _REQUEST is an array that contains variables passed in from the form
- This works on both PHP 4 and PHP 5

- PHP 4 allows you to access form variables directly by name, but this doesn't work in PHP 5

PHP 4 Only

```
<?
 header("Content-Type: text/html");
 print("<HTML><HEAD><TITLE>My Page</TITLE>");
 print("</HEAD>");   print("<BODY>");
 if ($_SERVER['REQUEST_METHOD'] == ‘POST') {
    print("foo = $foo, bar = $bar <P>");
 }
 print("<form method=post action=\"example.php\">");
 print("<input type=text name=\"foo\" value=\"zot\">");
 print("<input type=hidden name=\"bar\" value=3>");
 print("<input type=submit>");
 print("</form>");
 print("</BODY></HTML>");
?>
```

PHP 4 or PHP 5

```
<?
 header("Content-Type: text/html");
 print("<HTML><HEAD><TITLE>My Page</TITLE>");
 print("</HEAD>");   print("<BODY>");
 if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    print("foo = " . $_REQUEST["foo"] .
    ", bar = " . $_REQUEST["bar"] . "<P>");
 }
 print("<form method=post action=\"example2.php\">");
 print("<input type=text name=\"foo\" value=\"zot\">");
 print("<input type=hidden name=\"bar\" value=3>");
 print("<input type=submit>");
 print("</form>");
 print("</BODY></HTML>");
?>
```

More PHP

- Here we will focus on additional functions that will be helpful for you to complete the homework assignment
  - Random number generation, sort, arrays (previously covered)
  - Type Checking
    - is_array, is_string, is_long, is_double
  - Useful string functions
    - strlen, implode, explode, substr, strstr, trim, char access
  - File I/O
    - fopen, fread, feof, fclose, fwrite
  - Some examples

Type Checking

- PHP includes several functions to determine the type of a variable since it may not be obvious what the type is due to conversions

```
is_int($x)        // returns true if $x is an integer
is_double($x)     // returns true if $x is a double
is_array($x)      // returns true if $x is an array
is_string($x)     // returns true if $x is a string
is_null($x)       // returns true if $x is a null
```

String Functions

- We can access a string as an array to retrieve individual characters:

```
$s="hithere";
$z = $s[0] . $s[2] . $s[4];
print($z);                          // hte
```

- We can also assign characters to the string:

```
$s[2] = "F";
print($s);                          // hiFhere
```

Strings

- String length: strlen($s) returns the length of the string

```
$s="eat big macs";
for ($i=0; $i<(strlen($s)-1)/2; $i++) {
    $temp = $s[$i];
    $s[$i] = $s[strlen($s)-$i-1];
    $s[strlen($s)-$i-1] = $temp;
}
print($s);                          // Output : scam gib tae
```

Strings

- Substring: Searches a string for a substring

  Prototype:

  string **strstr** (string haystack, string needle)
  - Returns all of *haystack* from the first occurrence of *needle* to the end.
  - If *needle* is not found, returns **FALSE**.

```
$email = 'sterling@designmultimedia.com';
$domain = strstr ($email, '@');
print ($domain); // prints @designmultimedia.com
```

Strings

- strtolower($s) : returns $s in lowercase

```
$s="AbC";
$s = strtolower($s);          // $s = "abc"
```

- strtoupper($s) : returns $s in uppercase

```
$s = "AbC";
$s = strtoupper($s);          // $s = "ABC"
```

- trim($s) : returns $s with leading, trailing whitespace removed

```
$s = "    \n ABC   \r\n";
$s = trim($s);                          // $s = "ABC"
```

Trim is useful to remove CR's and Newlines when reading lines of data from text files or as input from a form (e.g. textbox, textarea)

Strings

- Substring: Format:

  string **substr** (string string, int start [, int length])
  - Substr returns the portion of *string* specified by the *start* and *length* parameters.
  - If *start* is positive, the returned string will start at the *start*'th position in *string*, counting from zero. For instance, in the string 'abcdef', the character at position 0 is 'a', the character at position 2 is 'c', and so forth.
- Examples:

  $rest = substr ("abcdef", 1); // returns "bcdef"

  $rest = substr ("abcdef", 1, 3); // returns "bcd"

Implode

- Implode is used to concatenate elements of an array into a single string

  string **implode** (string glue, array pieces)
  - Returns a string containing a string representation of all the array elements in the same order, with the glue string between each element.
- Examples

  $arr[]="A"; $arr[]="B"; $arr[]="C";

  $s = implode(",",$arr); // $s = “A,B,C”

  $s = implode("",$arr); // $s = “ABC”

Explode

- Explode is used to create an array out of a string with some delimiter

  array **explode** (string separator, string string)
  - Returns an array of strings, each of which is a substring of *string* formed by splitting it on boundaries formed by the string *separator*.
- Example

```
$s="eat:large:fries";
$arr = explode(":",$s);
print_r($arr);
print("<p>");
```

Output: Array ([0] => eat [1] => large [2] => fries)

File I/O

- Opening a file: fopen
- Format:

  int **fopen** (string filename, string mode)
  - Filename is the complete path to the file to open; must have proper permissions
  - Mode is one of the following
    - 'r' - Open for reading only; place the file pointer at the beginning of the file.
    - 'r+' - Open for reading and writing; place the file pointer at the beginning of the file.
    - 'w' - Open for writing only; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
    - 'w+' - Open for reading and writing; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
    - 'a' - Open for writing only; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
    - 'a+' - Open for reading and writing; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
  - Returns: a file pointer used to reference the open file

File I/O

- Reading from a text file:

  string **fgets** (int filepointer, int length)
  - Returns a string of up to length - 1 bytes read from the file pointed to by fp.
  - Reading ends when length - 1 bytes have been read, on a newline (which is included in the return value), or on EOF (whichever comes first).
  - We can use this function on files we have opened for reading

File I/O

- Writing to a text file:

  int **fwrite** (int fp, string string)
  - **fwrite()** writes the contents of *string* to the file stream pointed to by *fp*.
  - The file must be opened for writing

- Checking for end of file

  feof(int fp)

  Returns true if we have reached the end, false otherwise

- Closing a file

  fclose(int fp)

  Use when done with the file and close the file pointer

File I/O example

```
$fd = fopen ("/proc/cpuinfo", "r");
while (!feof ($fd)) {
    $oneline = fgets($fd, 4096);
    print("$oneline<br>");
}
fclose ($fd);
```

fgets

- IMPORTANT – Remember that fgets returns the string WITH the newline
- This is critical if you are going to perform comparisons
  - You'll get a false match if the newline is not accounted for
  - Easiest technique: trim out the newlines

```
$oneline = trim(fgets($fp, 1024));
```

Example

- Create a single PHP script that generates a form with a textarea
  - Allow the user to enter numbers in the textarea
  - Submit the form to the same script
  - Compute the sum of the numbers in the textarea and print it out

Example.php

```
<?php
        header("Content-Type: text/html");
        print("<HTML><HEAD><TITLE>My Page</TITLE>");
        print("</HEAD>");
        print("<BODY>");

        if($_SERVER['REQUEST_METHOD'] != "POST")
        {
                // We are loading for the first time,
                // not receiving a form.  So generate
                // a form allowing the user to enter
                // data in a text area and have it submitted
                // to this same script
                print("<FORM method=post action='example.php'>");
                print("Enter numbers below.<p>");
                print("<TEXTAREA name='myData' rows=10></TEXTAREA>");
                print("<INPUT type=submit>");
                print("</FORM>");
        }
```

Example.php

```
        else
        {
                // We are receiving data from our form
                // Put the text data into an array.  Each
                // is separated by a newline, so use explode
                // to parse
                $a = explode("\n",$_REQUEST['myData']);
                // Here we loop through and add up the numbers
                $total = 0;
                foreach ($a as $key=>$value) {
                        // Each element in the array is a string,
                        // but note that each will contain a \r
                        // whitespace at the end, so you may wish
                        // to trim these out. It is not really
                        // necessary in this example but you will
                        // normally want to trim just to be safe
                        $num = (int) trim($value);
                        $total += $num;
                }
                print("The sum of your numbers is $total<p>");
        }
 print("</BODY></HTML>");
?>
```

Accessing a MySql Database

- Here is the minimum for executing a mysql query from PHP.
- Given the following database:

```
mysql> describe data;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| username | varchar(255) | NO   | PRI | NULL    |       |
| val      | int(10)      | YES  |     | NULL    |       |
| password | varchar(255) | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+

mysql> select * from data;
+----------+------+----------+
| username | val  | password |
+----------+------+----------+
| david    |   31 | bjorn    |
| frank    |   95 | galileo  |
| kenrick  |   10 | uluru    |
| kirk     |   45 | jujube   |
+----------+------+----------+
```

Reading from the DB

```
// Database parameters
$db_location = "localhost";
$db_user_name = "test";
$db_password = "test";
$database_name = "test";

// Connect to the DB
$dbcnx = mysql_connect($db_location, $db_user_name, $db_password);
mysql_select_db($database_name);


// Display everything from the data table
$result = mysql_query("SELECT * FROM data;");
print("<table border=2>");
while ($row = mysql_fetch_assoc($result))
{
        $username = $row['username'];
        $val = $row['val'];
        $pw = $row['password'];
        print("<tr>");
        print("<td>$username</td> <td>$val</td> <td>$pw</td>");
        print("</tr>");
}
print("</table>");
?>
```

Writing to the DB

```
<?php
// Database parameters
$db_location = "localhost";
$db_user_name = "test";
$db_password = "test";
$database_name = "test";

// Connect to the DB
$dbcnx = mysql_connect($db_location, $db_user_name, $db_password);
mysql_select_db($database_name);

// Insert a new record
$result = mysql_query("INSERT INTO data (username, val, password) VALUES ('martin',55,'complexity');"
print("Result of insert: $result");  // True if successful

?>
```

SQL Injection Example

- PHP code

```
$id = $_REQUEST[“id”];
$pass = $_REQUEST[“password”];
$qry = “SELECT ccnum FROM cust WHERE id = $id AND pass=$pass”;
```

SQL Injection Example

- PHP code

```
$id = $_REQUEST[“id”];
$pass = $_REQUEST[“password”];
$qry = “SELECT ccnum FROM cust WHERE id = ‘$id’ AND
pass=‘$pass’”;
```

User inputs id of user to attack

For password, enters: **`' OR 1=1 –`**

-- is the comment operator, to ignore whatever comes afterwards

Another:

Password: **`'OR ''='`**

Update Counter

- What's wrong with this?

```
<?php
  $h = fopen("counter.txt","r");
  $count = fread($h, 100);
  fclose($h);
  $h = fopen("counter.txt","w");
  fwrite($h, $count + 1);
  fclose($h);
?>
```

Update Counter

- Better

```
<?php
  $h = fopen("counter.txt","r+");
  if (flock($h, LOCK_EX)) {
    $count = fread($h, 100);
    rewind($h);
    fwrite($h, $count + 1);
    fclose($h);
    flock($h, LOCK_UN);
?>
```

Session Variables

- Variables that keep their state from one PHP script to another
  - Generally stored as a cookie for the browsing session

```
<?php
  session_start();          // Must send before HTML


  $_SESSION['varname'] = value;  // Use isset to see if set
```

Summary

- PHP is an imperative language for the web
- Similarities to C, Java, and even interpreted languages such as Scheme
- Competition to ASP, .NET
- Can't do everything since server side only – often coupled with client-side languages such as JavaScript
- PHP version 5 not quite backward compatible with PHP 4
  - More OOP, references allow for more efficiencies
  - Highlights design choice of evolving language
- Easy to write sloppy code so one must be more disciplined in design of classes, functions, variables, HTML, documentation

Lots More to PHP

- We have only scratched the surface, but there is much more that PHP can do
  - Generate graphics (gd library)
  - Networking, Sockets, IRC, Email
  - LDAP
  - Regular Expressions
  - PDF
  - Java
  - XML
  - AJAX
  - Design methodologies (e.g. FuseBox, Smarty Templates, include files)
  - Many more
- See the excellent resources online
  - www.php.net
  - www.phpbuilder.com
  - www.zend.com