B336 Advanced Internet Computing

PHP Language

Learning Objectives

- Learn the basic construct available in the PHP language.
- Understand the features available in PHP to support web application development.

Lecture Outline

- PHP language syntax
- Support for web development
  - Form Handling
  - Cookies
  - Predefined web variables
- PHP and Databases

Server-side Scripting with PHP

- In *Topic 4: Internet Application Development - Programming, Scripting and Mark-up Languages,* we will compare PHP to the other prevalent server-side scripting technologies around.
- The purpose of this lecture is to give you an introduction to the language syntax, so you can begin doing some scripting work.

A simple PHP Script embedded in HTML

```
<html>
  <body>
    <? echo "<p>Hello World</p>" ?>
  </body>
</html>
```


Output

```
<html>
  <body>
    <p>Hello World</p>
  </body>
</html>
```

Basic Structure

- All PHP commands must be enclosed within a "<?php …?>" or "<? … ?>" tags.
  - Web servers can be configured to recognize PHP files in different ways - eg. by looking for file extension .php.
  - Web servers can be configured to run PHP files through the PHP interpreter.
  - When run through a PHP interpreter, the interpreter will process and replace the commands within these tags, and leave the rest of the file alone.
- We use “echo” or “print” to generate outputs.
- Almost every PHP command must end in a semi-colon (;).

A Slightly More Complicated Example

```
<html>
<body>
  <p>Our second PHP script</p>
  <?
    /* Comment: To print today's date */
    $today = date("Y-m-d");
    echo "<p>Today is: $today.</p>";
    # Another comment
  ?>
</body>
</html>
```


Comments

- Comments can be made within the script, and are enclosed within "/* … */" brackets, or by placing it after a "#" sign.
- All comments are ignored by the interpreter.

Output

```
<html>
<body>
  <p>Our second PHP script</p>
  <p>Today is: 2002-08-16.</p>
</body>
</html>
```


Variables

- All variables in PHP are prefixed with a dollar sign - including arrays and complex data types.
- Basic data types:

Type	Description
Integer	integer number
Double	floating point number
bool	Boolean (true or false)
Array	hybrid of ordered array and associative array
object	an object with properties and methods

Example Basic Variable Types

```
<?

$number = 5;
$string1 = "this is a string\n";
$string2 = 'this is another "string"';
$real = 37.2;

?>
```


Declaring and Using Data Types

- You do not have to formally declare the data type of a variable.
  - The type is inferred from it’s use.
  - But be careful with using variables of different types. Sometimes the automatic type conversions can change the type of the variables without warning.

Operators

- Eg.

Op	Meaning	Example Usage
>	Greater than	2 > 3
!=	Not equal	$x != 4
+	Addition	$x + 2 + $y
&&	Logical AND	($x > 4) && ($y < 3)
++	Increment by one	$y++ ;
.	Concatenation	$x . " and " . $y

etc...

Arrays

- There are two types of PHP arrays.
- Basic arrays
  - indexed by continuous numbers
  - our normal arrays
- Associative arrays
  - indexed by strings
  - indexed by non-continuous numbers.
  - Indexed by a mix of numbers and strings.

Implicit Array Indices

- Normal arrays indexed by 0, 1, 2, etc are declared without needing to specify the index:

```
<?
  $numbers = array(2, 3, 5, 7, 11);
  echo "$numbers[0] $numbers[4]" ;
?>
```

	[0]	[1]	[2]	[3]	[4]
$numbers	2	3	5	7	11

Explicit Array Indices

- Associative arrays needs explicit declaration on what the indices are:

```
<?
  $age = array("harry" => 32,
               "jane" => 11);
  echo $age["jane"] ;
?>
```

	["harry"]	["jane"]
$age	32	11

Assumed Array Indices

- Some array indices can be assumed from previous index numbers:

```
<?php
  $numbers = array(5 => 2, 3, 5);
  print "$numbers[5] $numbers[6]" ;
?>
```

	[5]	[6]	[7]
$numbers	2	3	5

Program Control: Conditionals

```
if ($a) {
    echo "a is true<BR>\n";
} elseif ($b) {
    echo "b is true<BR>\n";
} else {
    echo "neither a or b is true<BR>\n";
}
```

Program Control: Loops

```
do {
  $c = test_something();
} while ($c);
```

```
while ($d) {
  echo "ok<BR>\n";
  $d = test_something();
}
```

```
for ($i = 0; $i < 10; $i++) {
  echo "The value of i is $i<BR>\n";
}
```


Program Control: Loops

```
foreach ($num as $numbers) {
  echo "$num <BR>\n";
}
```


Splitting Control Blocks

- You can split one control statement over multiple <? … ?> tags. Eg. the following are equivalent

```
<? if ($a) { ?>
    <P>a is true.</P>
<? } else { ?>
    <P>a is not true.</P>
<? } ?>
```

```
<?
   if ($a) {
    echo "<P>a is true.</P>" ;
   } else {
    echo "<P>a is not true.</P>" ;
   }
?>
```


Splitting Control Blocks

- Splitting one control statement over multiple <? … ?> tags is useful when you have a large chunk of HTML to print out, and do not want the hassle of putting them through multiple echo statements.

Functions

- We can define and use functions:

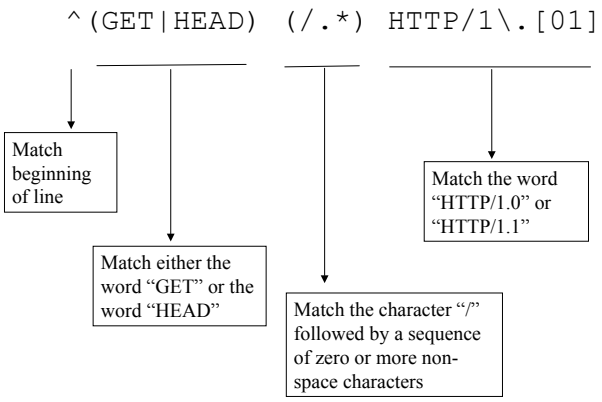
```
<?
  function add($a, $b) {
    return $a+$b ;
  }

  $value = add(2,3) ;
  print "<P>Two plus Three is $value.</P>" ;
?>
```


Regular Expressions (regexp)

- Regular expressions are general expressions we can use to match strings.
  - It is a very powerful and commonly used mechanism in programming for string manipulation.
- Eg. If I want to determine if $str matches a HTTP request line:

```
if (preg_match("!^(GET|HEAD) (/.*) HTTP/1\.[01]!", $str))
{
   …
}
```



Some Simpler Examples

```
# matches strings "harry" or "bob"
if (preg_match("/harry|bob/", $str)) {
  ...
}
```

```
# matches any lowercase character, followed by a
# space, followed by another lowercase character
if (preg_match("/[a-z]\w[a-z]/", $str)) {
  ...
}
```

```
# replace an uppercase letter in $str with "hello"
$new = preg_replace("/^[A-Z]/", "hello", $str) ;
```


Example File Handling

```
$fd = fopen ("data.txt", "r");        # open data.txt for reading
while (!feof ($fd)) {                 # while not reached end-of-file
   $buffer = fgets($fd);              # read one line into $buffer
   echo $buffer                       # do something with $buffer
}
fclose ($fd);                         # close the file
```

```
$fd = fopen ("data.txt", "a");        # open data.txt for appending
fputs($fd, "A new line.");            # write a line into the file
fclose ($fd);                         # close the file
```

```
$fd = fopen ("data.txt", "w");        # open data.txt for overwriting
fputs($fd, "A new line.");
fclose ($fd);
```


File Handling

- Files is the simplest method for storing persistent data (ie. data that does not disappear after the script finishes, eg values in variables)
  - There other more sophisticated ways (eg. databases, cookies, etc.) but they involve more overhead.
- PHP provides basic file-handling functions.

Processing HTML Forms

- Handling HTML forms and cookies is a basic mechanism in PHP.
- Any form input with a certain name will be available in the PHP script as a global variable of the same name.

Processing HTML Forms

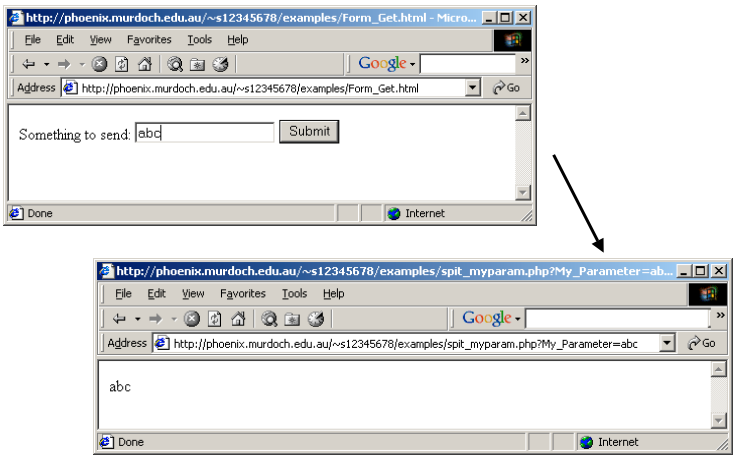
Form_Get.html

```
<html>
 <body>
  <form method="POST" action="spit_myparam.php">
   Something to send: <input type="text" name="My_Parameter">
   <input type="submit" value="Submit">
  </form>
 </body>
</html>
```

spit_myparam.php

```
<html>
 <body>
  <p>
   <? echo $My_Parameter ; ?>
  </p>
 </body>
</html>
```

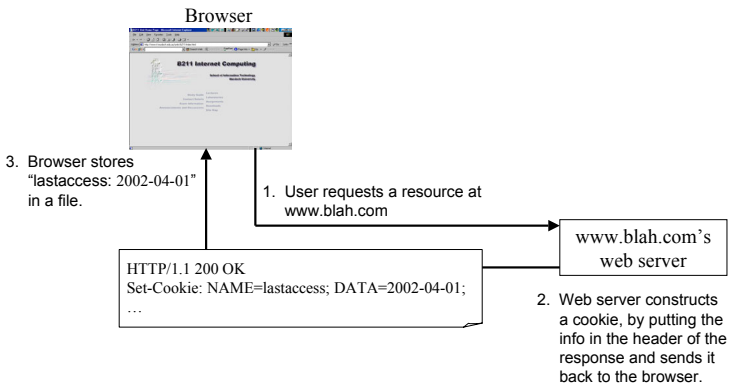
Note how the script have access to the form input element "My_Parameter" through a global variable $My_Parameter



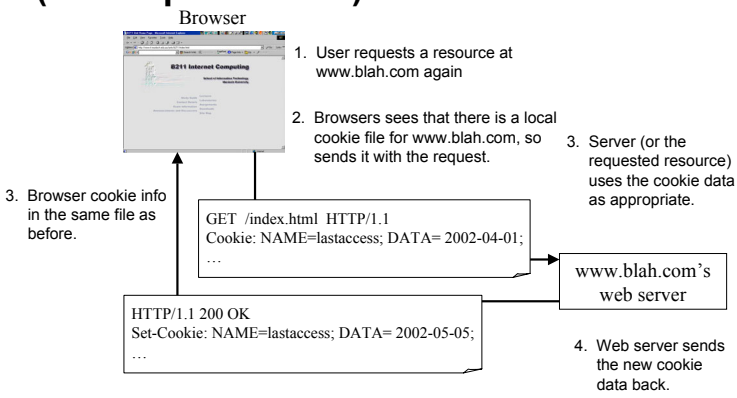
Support for Cookies

- PHP also has extensions to support manipulating HTTP Cookies.
- A review:
  - Cookies are sent by servers to clients (browsers), and are managed and stored locally by the clients. Servers requests such cookies at later sessions.

Example Cookie Transaction (first time)



Example Cookie Transaction (subsequent times)



The Web Application's Tasks for Handling Cookies

- The web application (at the web server side) has to
  - construct the cookie header in HTTP responses,
  - read the cookie header in HTTP requests,
  - process the data in the cookies.
- PHP extensions for cookies makes these steps transparent and easy to use.

Example: Creating a Cookie

```
<?  $today = date("d-m-Y H:i:s");
    setcookie ('lastaccess', $today, time()+3600); ?>

<html>
 <body>
  <? if ($lastaccess) {
       echo "<p>Welcome back. " ;
       echo "Your last visit was on $lastaccess.</p>" ;
     } else
       echo "<p>Welcome to this page for the first time.</p>" ;
  ?>
 <body>
</html>
```

The format of $today is "day-month-year hour:minutes:second". Eg "16-08-2002 14:36:07"

The cookie is set with the name "lastaccess", the value of $today, and expires 1 hour (3600 seconds) from when it is created.

testcookie.php

Predefined Variables

- PHP has a set of available predefined arrays containing variables from the web server (if applicable), the environment, and user input.

Example Predefined Variables

Variable Name	Description
$_COOKIE['lastaccess']	Cookie with name 'lastaccess' - same as our previous example
$_SERVER['QUERY_STRING']	The query string of the request, as described last lecture.
$_SERVER['SERVER_NAME']	The name of the web server running the PHP script.
$_SERVER['SCRIPT_NAME']	The name of the current PHP script.
$_GET['My_Parameter']	The value of a parameter called "My_Parameter", passed from a GET request
$_POST['My_Parameter']	The value of a parameter called "My_Parameter", passed from a POST request
$_ENV['LOGNAME']	The 'logname' environment variable.

Example: Using Predefined Variables

```
<?      # If the HTTP_USER_AGENT string contains "MSIE"
        if(strstr($_SERVER['HTTP_USER_AGENT'),"MSIE")) {
?>
        … HTML for Microsoft IE ...
<?      } else {        ?>
        … HTML for other browsers ...
<?      }               ?>
```


Support for Web Applications

- As you can see with the examples HTML forms, cookies and built-in variables, PHP is created to support a lot of the basic web communication mechanisms.

PHP and Databases

- PHP has strong support for database interfacing.
  - It has library modules (called PHP ***extensions***) to support many of the most popular database servers on the market, including MySQL, Oracle, Sybase, mSQL, Generic ODBC, and PostgreSQL,

PHP and MySQL

- PHP is most commonly associated with the MySQL package, and use in tandem to create dynamic data-driven web sites.
- The close association is main due to:
  - Both packages being freely available,
  - Support for both comes usually through the same developer communities, and
  - PHP has large support through its mysql extensions.

PHP and Databases

- Since databases is not a component of this unit, you will not be required to connect to databases using PHP.
  - We will store our data in files.
  - If you design your script code modularly, you should be able to change between using different data sources with minimal changes to code.
- The reason for mentioning databases here is to indicate it's importance, and to note that almost all server-side applications are connected to databases.

In the labs...

- This lecture is NOT a complete introduction to the PHP language.
  - It is only to give an overview of what is available in the language.
- You will deal with the details of the language as you do the lab exercises in labs week 7-10 by
  - Following examples scripts (such as from this week's lectures).
  - Referring to the textbook chapter 12.
  - Other example code.

References

- Main PHP site:
  - http://www.php.net/
  - A survey of other important PHP sites can be found there at http://www.php.net/sites.php and http://www.php.net/links.php.
- Tutorials:
  - http://www.zend.com/zend/art/intro.php
  - http://hotwired.lycos.com/webmonkey/programming/php/index.html
  - http://www.devshed.com/Server_Side/PHP/Introduction/page1.html

References

- Most of the example scripts used in this week's lectures are taken from the online tutorials above.