Cnt 4714: Enterprise Computing
Spring 2013

Introduction to PHP – Part 1

Instructor: Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cnt4714/spr2013

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida
Introduction to PHP

• We looked at a simple PHP example at the end of the set of notes that covered the installation of the Apache HTTP Server and PHP.

• PHP scripts can be created with any text editor, although Notepad++ is quite convenient for PHP scripting. I’ll primarily use it in the examples.

• PHP script files should be saved with a .php extension.

• When PHP is embedded inside XHTML documents, as it commonly is, several different delimiters can be used. These are illustrated on the next page.
Introduction to PHP

Common PHP delimiter in XHTMML documents where more that one type of embedded script might be used.
Introduction to PHP

Note: To ensure portable, reusable code, it is best to use the standard tags instead of the short or ASP-style tags for the simple reason that server configurations are unique - use the standard style because you know you can count on it as part of any configuration.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd>
<html xmlns="http://www.w3.org/1999/xhtml">
  
  <head>
    <title>Hello From PHP</title>
  </head>

  <body style="font-family: arial, sans-serif; background-color: #856363" background=image1.jpg>
    <h1>Hello From PHP</h1>

    <?php
      print "Current Information";
      phpInfo();
    ?>

  </body>

</html>

Standard PHP delimiter. If your PHP installation set-up has short_open_tag enabled, you can actually remove the php from the delimiter. If asp_tags are enabled you can use <% and %> as delimiters.
As with any programming language, good practice in writing scripts would require comments to be included within the script.

In-line comments in PHP are indicated with two forward slashes (//).

Comments can appear anywhere in the script file and can appear in any position on any line.

Multiple line comments are delimited with /* and */

Most PHP implementations also allow # to delimit in-line comments.
Variables In PHP

• You can select just about any set of characters for a variable name in PHP, but they must:
  – Use a dollar sign ($) as the first character.
  – Use a letter or an underscore character (_) as the second character.

• As with any programming/scripting language, good practice would suggest selecting variable names that help describe their function. For example $counter is more descriptive than $c or $ctr.

• You can use the echo statement or the print() function to output data in PHP. Which you use is more a matter of personal taste or style than anything else.
Variables In PHP

• To print out the value of a variable $x, write the following PHP statement:
  
  ```php
  print("$x");
  ```

• The following code will output “Candice is 26 years old”.
  
  ```php
  $age=26;
  print("Candice is $age years old.");
  ```

• The next page illustrates a full example using PHP variables.

Note: Constants are defined in PHP using the built-in `define()` function. As its name would imply a constant’s value cannot be changed once it is set.
```php
<?php

$firstNum = 12;
$secondNum = 365;
$temp = $firstNum;
$firstNum = $secondNum;
$secondNum = $temp;

print("first number = $firstNum <br /> second number = $secondNum");

?>
```

- first number = 365
- second number = 12
Data Types In PHP

• PHP is a dynamically typed language. This basically means that variables are not assigned a type when the variable is declared. Variable type is determined through assignment.

• The standard data types in PHP are shown in the table below:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>true</td>
<td>Either true or false</td>
</tr>
<tr>
<td>Integer</td>
<td>5</td>
<td>A whole number</td>
</tr>
<tr>
<td>Float or Double</td>
<td>3.14159</td>
<td>A floating-point number</td>
</tr>
<tr>
<td>String</td>
<td>“Hello”</td>
<td>A collection of characters</td>
</tr>
<tr>
<td>Object</td>
<td></td>
<td>An instance of a class</td>
</tr>
<tr>
<td>Array</td>
<td></td>
<td>An ordered set of keys and values</td>
</tr>
<tr>
<td>Resource</td>
<td></td>
<td>Reference to a 3rd party resource (e.g. a database)</td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td>An uninitialized variable</td>
</tr>
</tbody>
</table>
The concatenation operator in PHP is the period.
Data Types In PHP

• Technically speaking, there are two types of strings in PHP: parsed and unparsed.
• Parsed strings are defined using double quotes and are parsed by PHP.
• Unparsed strings are defined using single quotes and are taken as is (they are not parsed).
• What’s the difference? Within a parsed string, any references to variables within that string will be automatically replaced with their respective values, whereas within an unparsed string nothing is replaced.
• The example on the next page will clarify the differences.
```php
<?php

$myInt = 50;
$myString1 = "Hi there";
$myString2 = 'The value of $myInt = $myInt';
/* The next echo statement includes a reference to $myInt in a parsed string */
echo "The integer variable $myInt = " . $myInt . "<br />
/* The next echo statement includes a reference to $myInt in an unparsed string */
echo 'The integer variable $myInt = ' . $myInt . "<br />
/* The next echo statement prints an unparsed string with no variable references */
echo $myString1 . "<br />
/* The next echo statement prints an unparsed string with a variable reference */
echo $myString2 . "<br />
/* The next echo statement contains both parsed and unparsed variable references */
echo "The integer variable " . '$myInt' . " = " . $myInt . "<br />
?>

</body>
</html>
```
Parsed Versus Unparsed Strings In PHP

- The integer variable 50 = 50
- The integer variable $myInt = 50
- Hi there
- The value of $myInt = $myInt
- The integer variable $myInt = 50
Arithmetic Operations In PHP

- PHP supports all normal arithmetic operators, with the normal semantic associated with each operator.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Effect</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>$x = 2 + 2;</td>
<td>$x is assigned 4.</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>$y = 3; $y = $y - 1;</td>
<td>$y is assigned 2.</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>$y = 14 / 2;</td>
<td>$y is assigned 7.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>$z = 4; $y = $z * 4;</td>
<td>$y is assigned 16.</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>$y = 14 % 3;</td>
<td>$y is assigned 2.</td>
</tr>
</tbody>
</table>

- PHP supports automatic increment and decrement operations in both prefix and postfix form, i.e., -- and ++.

- Using an unassigned variable in an expression does not generate an error, the value is simply assumed to be null.

```php
<?php
    $y = 3;
    $y = $y + $x + 1;
    print("x=$x y=$y");
?>
```

The output is: x=y=4
String Variables In PHP

- PHP supports character string variables and this is a widely used aspect of PHP in handling form data.
- Be careful in PHP not to mix numeric and string types together in an expression.
- For example, you might expect the following statements to generate an error message, but they will not. Instead, they will output “y=1”.

```php
<?php
    $x = "banana";
    $sum = 1 + $x;
    print("y=$sum");
?>
```
String Variables In PHP

- The string concatenation operator in PHP is the period as shown below:

```php
<?php
    $firstname = "Megan";
    $lastname = "Fox"
    $fullname = $firstname . $lastname;
    print("Full name = $fullname");
?>
```

The output of this script would be: **Fullname=MeganFox**

You can also use double quotation marks to create concatenation directly. Using the above example you could do the following: `$fullname2 = "$firstname $lastname"`; This would have the same effect as: `$fullname2 = $firstname . $lastname;`
String Variables In PHP

- PHP supports a large variety of string handling functions. A few of the more commonly used ones are illustrated on the next few pages.

- Most string functions require you to send them one or more arguments.

- Arguments are input values that functions use in the processing they do.

- Often functions return a value to the script based on the input arguments. For example:

  `$len = strlen($name);`

  - **$len**: Variable or value to work with
  - **strlen($name)**: Name of function
  - Receives the number of characters in $name
String Variables In PHP

**strlen() function:**

- This function returns the number of characters in the string argument to the function. Consider the following script:

```php
<?php
$comments = "Good Job";
<len = strlen($comments);
print ("Length=$len");
?>
```

This PHP script would output "Length=8".
String Variables In PHP

**trim() function:**

- This function removes any blank characters from the beginning and end of a string. For example, consider the following script:

  ```php
  <?php
  $in_name = "      Megan  Fox  ";
  $name = trim($in_name);
  print ("name=$name\n$name");
  ?>
  
  This PHP script would output "name=Megan  FoxMegan  Fox".
String Variables In PHP

`strtolower()` and `strtoupper` functions:

- These functions return the argument string in all uppercase or all lowercase letters, respectively. For example, consider the following script:

```php
<?php
    $inquote = "Now Is The Time";
    $lower = strtolower($inquote);
    $upper = strtoupper($inquote);
    print("upper=$upper lower=$lower");
?>
```

This PHP script would output “upper=NOW IS THE TIME lower = now is the time”
String Variables In PHP

`substr()` function:

- This function enables a PHP script to extract a portion of the characters in a string variable. The general syntax is:

```php
$part = substr( $name, 0, 5);
```

Assign the extracted sub-string into this variable.

Starting position to start extraction from.

Extract from this string variable.

Number of characters to extract. (If omitted it will continue to extract until the end of the string.)
String Variables In PHP

**substr() function:**

- The `substr()` function enumerates character positions starting with 0 (not 1),
  - For example, in the string “Homer”, the “H” would be position 0, the “o” would be position 1, the “m” position 2, and so on.
- For example, the following would output “Month=12 Day=25”.

```php
<?php
    $date = "12/25/2002";
    $month = substr($date, 0, 2);
    $day = substr($date, 3, 2);
    print ("Month=$month Day=$day");
?>
```
String Variables In PHP

substr() function:

- This example does not include the third argument (and thus returns a substring from the starting position to the end of the search string).

```php
<?php
    $date = "12/25/2010";
    $year = substr($date, 6);
    print ("Year=$year");
?>
```

- The above script segment would output “Year=2010”.
Controlling Script Flow In PHP

- PHP contains the normal control statements that handle decision making and iteration within a script.
- Normal logical operators are all supported with their standard semantics.
- As with many modern programming and scripting languages remember to use `==` in a logical comparison operation and not `=`. The single equal sign is an assignment operator and as such is always true. No syntax error is generated.
- The table on the following page illustrates the common logical operators in PHP.

**Note:** PHP also contains a `===` logical comparison operator (called the identical operator). This binary operator returns true iff its two operands are equal in value and also have the same type.
# Controlling Script Flow In PHP

<table>
<thead>
<tr>
<th>Test Operator</th>
<th>Effect</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| `==`          | Equal to          | if ($x == 6) {
                      |     $x = $y + 1;
                      |     $y = $x + 1;
                     }                            | Run the second and third statements if the value of $x is equal to 6. |
| `!=`          | Not equal to      | if ($x != $y) {
                      |     $x = 5 + 1;
                     }                                | Run the second statement if the value of $x is not equal to the value of $y. |
| `<`           | Less than         | if ($x < 100) {
                      |     $y = 5;
                     }                                    | Run the second statement if the value of $x is less than 100.             |
| `>`           | Greater than      | if ($x > 51) {
                      |     print "OK";
                     }                                    | Run the second statement if the value of $x is greater than 51.           |
| `>=`          | Greater than or   | if (16 >= $x) {
                      | equal to                      |     print "x=$x";
                     }                                    | Run the second statement if 16 is greater than or equal to the value of $x. |
| `<=`          | Less than or      | if ($x <= $y) {
                      | equal to          |     print "y=$y";
                      |     print "x=$x";
                     }                                    | Run the second and third statements if the value of $x is less than or equal to the value of $y. |
Controlling Script Flow In PHP

- PHP includes several different forms of logical control statements (decision statements).

- The `if` statement has the form:

```php
if (expression) {
    //code to execute if expression evaluates to true
}
```

- The `if-else` statement has the form:

```php
if (expression) {
    //code to execute if expression evaluates to true
} else {
    //code to execute when expression evaluates to false
}
```
There is also an elseif clause that can be used with if statements for a nested stack of if statements. The basic syntax for this clause is:

```php
if (expression) {
    // code to execute if expression evaluates to true
} elseif (another expression) {
    // code to execute when expression evaluates to false
    // and another expression evaluates to true
} else {
    // code to execute if all expressions evaluate to false
}
```
Controlling Script Flow In PHP

- PHP includes a `switch` statement which allows for multiple options for a single evaluation of an expression. The basic syntax for the `switch` statement is:

```php
switch (expression) {
    case result1:
        //code to execute if expression evaluates to result1
        break;
    case result2:
        //code to execute if expression evaluates to result2
        break;
    . . .
    default:
        //code to execute if no break has been encountered
}
```
Controlling Script Flow In PHP

• The following example uses an input form (XHTML) and two values are extracted from the form (grade1 and grade2), passed to a PHP script which determines the average score, the maximum score and assigns a grade to the average for the student’s scores.

• We’ll get much more into forms and form handling in PHP later, but this simple example will illustrate several of the common threads that appear in form handling in PHP (and server side scripting in general).
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Grade Calculations</title>
  <meta http-equiv="content-type" content="text/html' charset=iso-8859-1" />
</head>
<body style = "font-family: arial, sans-serif; background-color: #856363" background=images.jpg>
  <form action="decisionsWithGlobals.php" method="post">
    <font size=4 color=blue>Please Enter Scores</font> <br />
    Enter First Score <input type="text" size="4" maxlength="7" name="grade1" /> <br /> Enter Second Score <input type="text" size="4" maxlength="7" name="grade2" /> <br />
    <input type="submit" value="Click To Submit" >
    <input type="reset" value="Clear And Restart" >
  </form>
</body>
</html>
Executing decisions.html
User enters two scores, clicks submit button.
Clicking the submit button triggers the action of the form and invokes the script `decisions.php`. The script generates this page. The PHP script is shown on the next page.
<?php

$grade1 = $_POST["grade1"];
$grade2 = $_POST["grade2"];
$average = ($grade1 + $grade2) / 2;
if ($average > 89) {
    print ("Average = $average You got an A");
} elseif ($average > 79) {
    print ("Average = $average You got a B");
} elseif ($average > 69) {
    print ("Average = $average You got a C");
} elseif ($average > 59) {
    print ("Average = $average You got a D");
} elseif ($average >= 0) {
    print ("Average = $average You got an F ");
} else {
    print ("Illegal average less than 0: Average = $average");
}
$max=$grade1;
if ($grade1 > $grade2) {
    $max = $grade1;
} else {
    $max = $grade2;
}
print ("<br /> Your maximum score was $max");
?>
</body>
</html>
Controlling Script Flow In PHP

• PHP supports three types of iterative constructs:
  – the while loop (both top and bottom tested versions are supported)
  – the for loop
  – and the foreach loop.

• The for and while loops act as you would expect given your knowledge of other programming languages. The foreach loop applies specifically to arrays in PHP. We’ll look at the foreach loop later.

• The next couple of pages show the basic syntax for each of the iterative constructs in PHP.
Controlling Script Flow In PHP

• The syntax for the top tested version of the `while` loop is:

```php
while (expression) {
    //statements to execute
}
```

• The syntax for the bottom tested version of the `while` loop is:

```php
do {
    //statements to execute
} while (expression);
```
Controlling Script Flow In PHP

• The basic syntax for the for statement is:

```php
for (initialization expr; test expr; modifying expr) {
    //statements to be executed
}
```

• The next couple of pages illustrates some of the nuances of dealing with counted loops in PHP.
Controlling Script Flow In PHP

```php
<?php
    define("LIMIT", 10);
    $counter = -4;
    for ($counter = 1; $counter <= LIMIT; $counter++) {
        $temp = 4000/$counter;
        echo "4000 divided by ". $counter. " is...". $temp. "<br />
    }
?>
</html>
```
What do you expect will happen when $counter == 0?
Division by zero is not a fatal error in PHP. Instead a warning is generated and execution continues. (Note that this warning is only displayed if `display_errors = On` in your `php.ini` file. Otherwise it looks like the following page.

The possible fixes are shown on the next two pages.
If `display_errors = Off` in your php.ini file (which is the typical default setting), you’ll see this page.

The possible fixes are shown on the next two pages.

For Loop Without A Break Statement:

4000 divided by -4 is...-1000
4000 divided by -3 is...-1333.333333333
4000 divided by -2 is...-2000
4000 divided by -1 is...-4000
4000 divided by 0 is...
4000 divided by 1 is...4000
4000 divided by 2 is...2000
4000 divided by 3 is...1333.333333333
4000 divided by 4 is...1000
4000 divided by 5 is...800
4000 divided by 6 is...666.6666666667
4000 divided by 7 is...571.42857142857
4000 divided by 8 is...500
4000 divided by 9 is...444.4444444444
4000 divided by 10 is...400
Use a break statement to terminate the loop in a division by zero case.
Use a continue statement to skip the division by zero case.
```php
<?php

define("LOWER_LIMIT", 1);
define("UPPER_LIMIT", 12);

for ($i = LOWER_LIMIT; $i <= UPPER_LIMIT; $i++) {
    echo "<tr> \n";
    for ($j = LOWER_LIMIT; $j <= UPPER_LIMIT; $j++) {
        echo "<td style="border: 1px solid black; width:25px; padding:4px; text-align:center;">";
        echo ($i * $j);
        echo "</td> \n";
    }
    echo "</tr> \n";
}

echo "</table?>";

?>
</html>
```

For Loop With A Continue Statement

Nested Loops
### Nested Loops

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
<td>30</td>
<td>33</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
<td>40</td>
<td>44</td>
<td>48</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
<td>50</td>
<td>55</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
<td>60</td>
<td>66</td>
<td>72</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
<td>70</td>
<td>77</td>
<td>84</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
<td>80</td>
<td>88</td>
<td>96</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
<td>90</td>
<td>99</td>
<td>108</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>110</td>
<td>120</td>
</tr>
<tr>
<td>11</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
<td>88</td>
<td>99</td>
<td>110</td>
<td>121</td>
<td>132</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
<td>60</td>
<td>72</td>
<td>84</td>
<td>96</td>
<td>108</td>
<td>120</td>
<td>132</td>
<td>144</td>
</tr>
</tbody>
</table>
• The example on the next couple of pages illustrates a while loop. Again, I’ve used a form to extract user input. This time the user input sets the lower and upper limit on the loop.
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" 
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>While Loop Demo</title>
<meta http-equiv="content-type" content="text/html" charset=utf-8" /><!--
</head>
<body style = "font-family: arial, sans-serif; 
background-color: #856363" background=imagel.jpg>
<form action="whileloop.php" method="post">
Select Starting Number
<select name="start">
<option>0</option> <option>1</option> <option>2</option>
<option>3</option> <option>4</option> <option>5</option> <option>6</option>
<option>7</option> <option>8</option> <option>9</option>
</select> <br />
Select Ending Number
<select name="end">
<option>0</option> <option>10</option> <option>11</option>
<option>12</option> <option>13</option> <option>14</option> <option>15</option>
<option>16</option> <option>17</option> <option>18</option> <option>19</option>
</select> <br />
<input type="submit" value="Submit" />
<input type="reset" value="Clear And Restart" />
</form>
</body>
</html>
Executing \texttt{whileloop.html}

User enters lower and upper limits, clicks submit button.
Controlling Script Flow In PHP

Clicking the submit button triggers the action of the form and invokes the script `whileloop.php`. The script generates this page. The PHP script is shown on the next page.

<table>
<thead>
<tr>
<th>Number</th>
<th>Square</th>
<th>Cube</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>125</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>216</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>343</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>512</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>729</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>121</td>
<td>1331</td>
</tr>
<tr>
<td>12</td>
<td>144</td>
<td>1728</td>
</tr>
</tbody>
</table>
```php
while ($i <= $end) {
    $square = $i * $i;
    $cube = $i * $i * $i;
    print ("<tr><td>$i</td><td>$square</td><td>$cube</td></tr>");
    $i++;
}
```

Embedding Control Structures

• Now that we’ve seen most of the control structures in PHP, we need to see how these control structures can be used more effectively to produce XHTML elements (or any other output).

• PHP is an embedded language that enables you to code both your XHTML and the supporting script in the same document.

• PHP takes this concept a bit further by allowing you to “turn off” the PHP parser during a control structure and embed non-PHP output without losing the logic provided by the control structure.

• The following example, illustrates this concept by displaying an image in your XHTML document only when a variable is set to true.
Embedding Control Structures

```php
<?php
    $displayKart = true;
    $displayBike = false;
    if ($displayKart) {
        echo "<img src=" sprint kart.jpg" >";
    }
    if ($displayBike) {
        echo "<img src="/Eddy Merckx.jpg" >";
    }
?>
```

Embedding Control Structures - Version 1
Embedding Control Structures

• Although the previous solution works and for novice PHP programmers it seems to be the most obvious technique, PHP provides an alternate syntax that is actually allows the embedding of the control structure into the markup.

• This alternative syntax is:

```php
<?php
    if (conditional): ?>
    - text/whatever that should be output but not parsed
    <?php endif;
?>
```

• This is shown in the next version of this example on the following page.
<?php
    $displayKart = true;
    $displayBike = false;
    if ($displayKart):
        <img src="sprint kart.jpg">
    endif;
    if ($displayBike):
        <img src="Eddy Merckx.jpg">
    endif;
</html>
Using PHP With REGISTERGLOBALS OFF

• Since PHP 4.2.0, PHP is shipped with the REGISTERGLOBALS configuration variable set to OFF. Prior to version 4.2.0 this variable was set to ON, but represented a fairly large security concern, so since that time the default setting is OFF. While this setting can be overridden by local system administrators, it is wise not to do so.

• When PHP is configured with REGISTERGLOBALS set to OFF, you need an extra step to receive input from forms, cookies, or session variables.

• You can tell your PHP site’s status of REGISTERGLOBALS by running the phpInfo() function that was shown in the hello.php script in the setting up PHP section of notes (repeated here)…
Create this file named `hello.php` and save it to the `htdocs` folder in the Apache server. Then start your browser and enter the URL: `http://localhost:8081/hello.php` and you should see output similar to that shown on the next slide.
Scan down the output listing until you get to the Core settings for PHP and in this table you'll find the setting for REGISTER_GLOBALS. First column is the local value and the second column is the master value (originally set implementation value).
When REGISTER_GLOBALS is set to OFF you must receive XHTML form input data using the $_POST, or $_GET associative arrays.

Go back and look at the PHP scripts on pages 46 and 49 and you will see the $_POST associative array has been used in both examples to extract the input form data.

We’ll deal with this in more detail later, but for now these two examples should give you a good idea of how form data extraction is handled in PHP scripts.

There are many other associative arrays utilized in PHP. The remainder of this set of notes is devoted to some of these arrays.
A More Complete Example Using Form Data

• Several of the previous examples illustrated extracting user input from form variables. Let’s look at one more example, where we prompt a user for their first and last name, and then use that information to display on every page of our website.

• First, let’s write the HTML form and PHP script necessary to get the user’s name.

• The XHTML document is shown on page 60 and the PHP script is shown on page 61, with an example shown on page 62.
A More Complete Example Using Form Data

The XHTML document:

```xml
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Extracting User Input From A Form</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>
<body style="font-family: arial, sans-serif; background-color: #856363" background-image1.jpg>
<form action="welcomeMessage.php" method="post">
    <font size=4 color=blue>Please Enter Your First And Last Name</font> <br>
    <label> Your Name </label>
    <input type="text" size="34" maxlength="37" name="Name" />
    <br>
    <input type="submit" value="Click To Submit" />
    <input type="reset" value="Clear And Restart" />
</form>
</body>
</html>
```
A More Complete Example Using Form Data

```php
<?php
    $Name = $_POST['Name'];
    echo("Welcome to our Website, $Name!\n");
?>
```

The PHP script
A More Complete Example Using Form Data

Please Enter Your First And Last Name

Your Name: Candice Swanepoel

Click To Submit  Clear And Restart

Welcome to our Website, Candice Swanepoel!
A More Complete Example Using Form Data

• Now suppose that we wanted to construct our website so that it showed the visitor’s name at the top of every page.

• While the script we’ve just constructed gets us the required data, there are a couple of problems we need to overcome, if we’re going to do this right.

  1. We need the name at the top of every page, not just the first one.
  2. We have no control over which page at our site the user might first enter.

• The first problem is fairly easy to handle, once the visitor’s name is in a variable on one page, we simply pass it as part of any request for another page by adding the name to the query string of all links.
A More Complete Example Using Form Data

• To pass the name as part of a query string of a link, we’ll embed PHP code right into the middle of an XHTML tag.

• This is perfectly legal and will work just fine.

• The revised link would look like the following:

```html
<a href="newpage.php?name=<?php echo urlencode($_GET["name"]); ?>" />
```

• The `urlencode` function takes special characters in the string, e.g., spaces, and converts them into the special codes they need to be in order to appear in the query string.

• For example, if the `$name` variable had a value of “Mark Llewellyn”, then, since spaces are not allowed in a query string, the output of `urlencode` would be “Mark+Llewellyn”. PHP would then convert it back automatically when it created the `$GET` variable in `newpage.php`.

A More Complete Example Using Form Data

• Now that we know how to handle the first problem, we now need to figure out how to get the name in the first place.

• We constructed a special XHTML document that contained a form for the visitor to enter their name. The problem is that we don’t want to force the user to enter our site by that page every time they visit the site.

• The solution is to have every page in the site check to see if a name has been specified and prompt the user for one if necessary.

• Pages in websites that can decide whether to display one thing or another are referred to as multipurpose pages. A basic template for a multipurpose page is shown on the following slide. With examples for our specific case shown on the pages following the template.
A More Complete Example Using Form Data

Template for a multipurpose webpage

Enter PHP mode

Exit PHP mode
A More Complete Example Using Form Data

Template for a multipurpose webpage – Alternate syntax format
Viewing Client/Server Environment Variables

- Knowledge of a client’s execution environment is useful to system administrators who want to provide client-specific information.

- Environment variables contain information about a script’s environment, such as the client’s web browser, the HTTP host and the HTTP connection.

  - The table on the next page summarizes some of the superglobal arrays defined by PHP.

- The XHTML document on page 70 displays the values of the server’s environment variables in a table. PHP stores the server variables and their values in the $_SERVER array. Iterating through the array allows one to view all of the server’s environment variables.
## Some Superglobal Environment Arrays

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$_SERVER</td>
<td>Data about the currently running server.</td>
</tr>
<tr>
<td>$_ENV</td>
<td>Data about the client’s environment.</td>
</tr>
<tr>
<td>$_GET</td>
<td>Data posted to the server by the <code>get</code> method.</td>
</tr>
<tr>
<td>$_POST</td>
<td>Data posted to the server by the <code>post</code> method.</td>
</tr>
<tr>
<td>$_COOKIE</td>
<td>Data contained in cookies on the client’s computer.</td>
</tr>
<tr>
<td>$GLOBALS</td>
<td>Array containing all global variables.</td>
</tr>
</tbody>
</table>
Iterate through the $_SERVER array to list all of the SERVER variables for the current server on which PHP is running.