INTRODUCTION TO WEB PROGRAMMING II

A

LECTURE NOTE

PREPARED

BY

MUHAMMAD SALISU ALI

FEDERAL UNIVERSITY DUTSE

16 APRIL, 2021

How to work with form data

- **How to get data from text boxes, password boxes, and hidden fields**
  - A text box allows the user to type data into the box.
  - A password box allows the user to type data into the box, but it obscures the characters typed by the user. This prevents *sensitive data* from being displayed on the screen.
  - A hidden field allows the programmer to add additional name/value pairs to the form.
  - If the form uses the GET method, the data for the fields is displayed in the URL.
- **Attributes of the HTML <input> tag for getting text input:**
  - **type** - Set to “text” for a text field, “password” for a password field, or “hidden” for a hidden field. The default is “text”. In addition, there are more HTML5 types including “email” for an email address, “url” for a URL, and “tel” for a telephone number.
  - **Name** - The name for the field when the form is submitted.
  - **Value** - The value for the field. For hidden fields, this attribute must be set for the field to work correctly.

- The URL when using the GET method

  process_data.php?user_name=rharris&password=s3cr3t72&action=login

- The PHP for the GET method

```
<?php
        $user_name = filter_input(INPUT_GET, 'user_name');
        $password = filter_input(INPUT_GET, 'password');
        $action = filter_input(INPUT_GET, 'action');
?>
```

- The URL when using the POST method

  process_data.php

- The PHP for the POST method

```
<?php
        $user_name = filter_input(INPUT_POST, 'user_name');
        $password = filter_input(INPUT_POST, 'password');
        $action = filter_input(INPUT_POST, 'action');
?>
```

How to get data from a radio button

- A radio button allows the user to select one option from a group of options.
- If none of the radio buttons in the group are selected, the name/value pair for the group is not submitted to the server as part of the form, and the filter_input function returns NULL.
- If you code the checked attribute for more than one radio button, the browser selects the last radio button when the page loads.
- Attributes of the HTML <input> tag for radio buttons:
  - **type** - Set to “radio” for a radio button.
  - **name** - The same name must be used for all radio buttons in a group.
  - **value** - The value for a radio button. This attribute is necessary for a radio button to work correctly.
  - **checked** - This optional attribute selects the default radio button.
- The HTML for three radio buttons in a group
  - <input type="radio" name="card_type" value="visa" checked> Visa
  - <input type="radio" name="card_type" value="mastercard"> MasterCard
  - <input type="radio" name="card_type" value="discover"> Discover

- The PHP to access a radio button group

```
<?php
        $card_type = filter_input(INPUT_POST, 'card_type');
?>
```

- The PHP to add a default value for a group with no default button

```
<?php
        $card_type = filter_input(INPUT_POST, 'card_type');
        if ($card_type == NULL) {
                $card_type = 'unknown';
        }
?>
```

How to get data from a check box

- A check box allows the user to select an option.
- The **isset** function returns TRUE if the check box is selected and FALSE if it isn't.
- Attributes of the HTML <input> tag for check boxes:
  - **type** - Set to "checkbox" for a check box.
  - **name** – Different names must be used for each check box. However, if you want to define an array of check boxes, you can use the same name as shown below.
  - **checked** - Text used when the check box is selected. This attribute is only necessary if you are working with an array of check boxes as below.
- The HTML for three check boxes

```
<input type="checkbox" name="pep" checked> Pepperoni<br>
<input type="checkbox" name="msh"> Mushrooms<br>
<input type="checkbox" name="olv"> Olives
```

- The PHP to access the check box data

```
<?php
    $pepperoni = isset($_POST['pep']);
    $mushrooms = isset($_POST['msh']);
    $olives = isset($_POST['olv']);
?>
```

How to get data from an array of check boxes

- If a check box name ends with [], PHP adds the check box to an array that's nested in the $_GET or $_POST array. This allows multiple values to be submitted with the same name.

- If none of the check boxes in an array of check boxes are selected, the array won't be set in the $_GET or $_POST array.
- If you're going to use the checkbox values, you should use a PHP function like the filter_input function to get the values of the array and to make sure they are safe. To do that, you can use filters like the ones shown in the next slide.
- The HTML that stores three related check boxes in an array

```
<input type="checkbox" name="top[]" value="pep"> Pepperoni<br>

<input type="checkbox" name="top[]" value="msh"> Mushrooms<br>

<input type="checkbox" name="top[]" value="olv"> Olives
```

- PHP that accesses the array and its values

```
<?php
    $toppings = filter_input(INPUT_POST, 'top', FILTER_SANITIZE_SPECIAL_CHARS,
                             FILTER_REQUIRE_ARRAY);
    if ($toppings !== NULL) {
        $top1 = $toppings[0];
        $top2 = $toppings[1];
        $top3 = $toppings[2];
    }
?>
```

- PHP that uses a loop to process the array

```
<?php
    $toppings = filter_input(INPUT_POST, 'top', FILTER_SANITIZE_SPECIAL_CHARS,
    FILTER_REQUIRE_ARRAY);
    if ($toppings !== NULL) {
        foreach($toppings as $key => $value) {
        echo $key. ' = ' . $value . '<br>';
      } else {
        echo 'No toppings selected.';
      }
?>
```

How to get data from a drop-down list

- A drop-down list lets the user select one option from a group of options.
- If the selected attribute isn't coded, the first option in the drop-down list is selected by default.
- Attributes of the HTML <select> tag for drop-down lists

  name - The name for the drop-down list.
- Attributes of the HTML <option> tag

  value - The value for the option.

  selected - This optional attribute selects the option.
- The HTML for a drop-down list

```
<select name="card_type">
        <option value="visa">Visa</option>
        <option value="mastercard">MasterCard</option>
        <option value="discover">Discover</option>
</select>
```

- The HTML to set a default option

```
<option value="mastercard" selected>MasterCard</option>
```

- The PHP to access the drop-down list data

```
<?php $card_type = filter_input(INPUT_POST, 'card_type');?>
```

How to get data from a list box

- A list box works similarly to a drop-down list. However, you set the size attribute of the <select> tag to the number of options that you want displayed in the list box.
- A list box allows the user to select zero or more options from a group of options.
- If a list box allows multiple options to be selected, the list box name must end with [].
- Attributes of the HTML <select> tag for list boxes:

  size - When set to 1 or omitted, a drop-down list displays the options. When set to 2 or more, a list box displays the specified number of options.

  multiple - This optional attribute allows the user to select multiple options in the list box. It should only be used if the size attribute has been set to a value of 2 or more.

- The HTML for a list box that allows multiple options to be selected

```
<select name="top[]" size="3" multiple>
    <option value="pep" selected>Pepperoni</option>
    <option value="msh">Mushrooms</option>
    <option value="olv">Olives</option>
</select>
```

- The PHP for a list box that allows multiple options to be selected

```
<?php
    $toppings = filter_input(INPUT_POST, 'top', FILTER_SANITIZE_SPECIAL_CHARS,
    FILTER_REQUIRE_ARRAY);
    if ($toppings !== NULL) {
        foreach ($toppings as $key => $value) {
        echo $key. ' = ' . $value . '<br>';     // '0 = pep' and '1 = msh‘
       }
    } else {
        echo 'No toppings selected.';
    }
?>
```

How to get data from a text area

- A text area lets the user enter multiple lines of text. To create a text box, you use the <textarea> tag.
- To set the default text for a text area, code the default text between the opening and closing <textarea> tags.
- If the user types past the end of the line, a text area uses a soft return to start a new line. If the users presses the Enter or Return key, a text area uses a hard return to start a new line.
- If the user doesn't enter any text, the name/value pair for the text area is submitted to the server with an empty string for the value.
- Attributes of an HTML <textarea> tag:

  name - The name for the text area.

  rows - The approximate number of lines for the text area.

  cols - The approximate number of characters for each line in the text area.
- The HTML for a text area

```
<textarea name="comment" rows="4" cols="50">
        Welcome to PHP and MySQL!
</textarea>
```

- The URL when using the GET method When the user includes spaces in the text area

  process_data.php?comment=Welcome+to+PHP+and+MySQL!
- When the user presses the Enter or Return key to start a new line

  process_data.php?comment=Welcome+to%0D%0APHP+and+MySQL!
- When the user doesn’t enter any text

  process_data.php?comment=
- The PHP to get the data from the text area

  <?php $comment = filter_input(INPUT_POST, 'comment');?>

How to display data on a web page

- How to format special characters
- An HTML character entity lets you display some special characters on a web page.
- The htmlspecialchars function converts some special characters into character entities. This provides a way to display special characters and helps to guard against XSS attacks.
- Syntax of the htmlspecialchars function

  htmlspecialchars($string[, $quote_style[, $charset[, $double_encode]]]**)**

  - $string - The string to convert. This parameter is required.
  - $quote_style - Specifies how to convert single and double quotes to their HTML entities. The ENT_COMPAT constant, which is the default, only converts double quotes. The ENT_QUOTES constant converts single and double quotes. The ENT_NOQUOTES constant doesn't convert single or double quotes.
  - $charset - Specifies the character set of the $string parameter. The default is "ISO-8859-1".
  - $- A Boolean value that specifies whether to double encode charadouble_encode cter entities. The default is TRUE. As a result, by default, character entities are double encoded.

- Common HTML character entities

Character	Character entity	Character	Character entity
&	&amp;	"	&quot;
<	&lt;	'	&#039;
>	&gt;	Non-breaking space	

- A double-encoded less than (<) entity
  <
- PHP that converts special characters to character entities

```
<?php
        $comment = $_POST['comment'];
        $comment = htmlspecialchars($comment);
?>
<p><?php echo $comment; ?></p>
```

- statement that prevents double encoding

```
$comment = htmlspecialchars($comment, ENT_COMPAT, 'ISO-8859-1', false);
```

How to format line breaks

- The **nl2br** function converts new line characters in a string to HTML <br> tags. This lets you display the line breaks on a web page.
- Syntax of the nl2br function

  nl2br($string[, $is_xhtml]**)**
- Parameters of the nl2br function:
  - $string - The string to convert. This parameter is required.
  - $is_xhtml - A Boolean value that indicates whether to use the XHTML syntax (<br />) or the HTML syntax (<br>) for a line break. The default is TRUE.
- PHP that converts line break characters to HTML line break tags

```
<?php
    $comment = filter_input(INPUT_POST, 'comment');
    $comment = nl2br($comment, false);      // use <br> tags, not <br /> tags
?>
<p><?php echo $comment; ?></p>
```

How to display data with echo and print statements

- The **echo** and **print** statements send strings to the web page. Non-string values are converted to strings before they are sent to the web page.
- The echo statement can accept *one* or *more* string values, but the print statement can only accept *one* value.
- The parentheses are optional. However, if you are providing multiple values to the echo statement, you must omit the parentheses.
- The echo statement doesn't return a value and cannot be used as part of an expression. However, the print statement returns a value of 1 so it can be used as part of an expression.
- The echo and print statements aren't functions. They are part of the PHP language definition.

- The echo statement

  Syntax

  echo $var1

  echo($var1)

  echo $var1 [, $var2 ...]

- Examples

  echo 'Welcome to PHP and MySQL!';

  echo 'Name: ' . $name;

  echo('Name: ' . $name);

  echo 'Cost: $', $cost;

- The print statement

  Syntax

  print $var1

  print($var1)

- Examples

  print 'Welcome to PHP and MySQL!';

  print 'Name: ' . $name;

  print('Name: ' . $name);

  Using print in an expression

  <?php ($age >= 18) ? print('Can vote.') : print('Cannot vote.');?>

How to Code Conditional Expressions

- How to use the equality and identity operators
- For simple comparisons, the two equality operators are sufficient. If, for example, you want to test whether a numeric variable contains a certain number, the equal operator works just fine.
- When the tests are more complex, however, unexpected results may occur when you use the equality operators.
- The problem is that the equality operators perform type coercion. This means that if different types of data are being compared, the values are converted to the same data type before the comparison takes place.
- The equality operators perform type coercion. Type coercion converts data from one type to another. PHP follows the rules shown below when performing type coercion.
- When converting to the Boolean type, the following values are equivalent to FALSE: NULL, 0, 0.0, “0”, an empty string, and an empty array. All other values are equivalent to TRUE.
- The identity operators do not perform type coercion. If the two operands are of different types, the result is always FALSE. As a result, all of the expressions listed above for the equality operator would return FALSE for the identity operator.

- The equality operators

==	Equal	$last_name == 'Harris'
!=	Not equal	$months != 0
<>	Not equal	$months <> 0

- PHP Type Coercion Rules

Operand 1	Operand 2	Action
NULL	String	Convert NULL to an empty string and compare as two strings.
Boolean or NULL	- Not a string	- Convert both to Boolean and compare.
String	- Number	- Convert string to a number and compare as two numbers.
Numeric string	- Numeric string	- Convert strings to numbers and compare as two numbers.
Text string	- Text string	- Compare strings as if using the **strcmp** function.

- The identity operators

===	Equal	$last_name === 'Harris'
!==	Not equal	$months !== 0

- Unusual results with the equality operator

Expression	Result	Description
null == ''	true	NULL is converted to the empty string.
null == false	true	NULL is converted to FALSE.
null == 0	true	NULL is equal to any value that evaluates to FALSE.
false == '0'	true	Empty strings and “0” are converted to FALSE.
true == 'false'	true	All other strings are converted to true.
3.5 == "\t3.5 mi”	true	The string is converted to a number and then compared.
INF == 'INF'	false	The string “INF” is converted to 0.
0 == ''	true	The empty string is converted to 0.
0 == 'harris'	true	Any string that is not numeric is converted to 0.

How to use relational operators

- When you use the relational operators, the operands are converted to the same types as described in the previous figure.
- When both operands are strings, they are compared character by character from the start of the strings based on the ASCII value of each character.
- The relational operators

<	Less than	$age < 18
<=	Less than or equal	$investment <= 0
>	Greater than	$test_score > 100
>=	Greater than or equal	$rate / 100 >= 0.1

- Comparing strings to numbers with the relational operators

1 < '3' true The string “3” is converted to the number 3.

'10' < 3 false The string “10” is converted to the number 10.

- Comparing strings with the relational operators

'apple' < 'orange'	true	An earlier letter is less than a later letter.
'apple' < 'appletree‘	true	When characters are the same, shorter strings are less than longer strings.
'Orange' < 'apple'	true	A capital letter is less than a lowercase letter.
'@' < '$'	false	Other characters are compared using their ASCII value.

- Unusual results with the relational operators

0 <= 'test'	true	The string "test" is converted to 0.
‘ ' < 5	true	The empty string is converted to 0.
false < true	true	FALSE is considered less than TRUE.
null < true	true	NULL is converted to FALSE.

How to use the logical operators

- Parentheses have the highest order of precedence so the operations within parentheses are done first, working from the innermost sets of parentheses to the outermost sets.
- To clarify the way an expression is evaluated, you should use parentheses.
- You can use arithmetic expressions within a conditional expression. In the order of precedence, the arithmetic operators come between the NOT operator and the relational operators.
- The logical operators

  ! NOT Returns the opposite Boolean value of its expression.

  && AND Returns TRUE only when the expressions on both sides are TRUE.

  || OR Returns TRUE when the expression on either side or both sides is TRUE.
- The logical operators in compound conditional expressions

  The NOT operator

  !is_numeric($number)

  The AND operator

  $age >= 18 && $score >= 680

  The OR operator

  $state == 'CA' || $state == 'NC'

- The order of precedence for operators in conditional expressions

Order	Operators	Description
1	!	The NOT operator
2	<, <=, >, >=, <>	Relational operators
3	==, !=, ===, !==	Equality and identity operators
4	&&	The AND operator
5	||	The OR operator

- The logical operators in complex conditional expressions
  - AND and OR operators

```
$age >= 18 && $score >= 680 || $state == 'NC‘
```

  - AND, OR, and NOT operators

```
!$old_customer || $loan_amount >= 10000 && $score < $min_score + 200
```

  - How parentheses can change the evaluation

```
(!$old_customer || $loan_amount >= 10000) && $score < $min_score + 200
```

How to code the selection structures

- **How to code if statements with else clauses**
- If you only have one statement after an if statement or an else clause, you don't have to put braces around that statement.
- You can nest an if statement within the if, else if, or else clause of another if statement. This nesting can continue many layers deep.
  - An if clause with one statement and no braces

```
if (!isset($rate)) $rate = 0.075;
```

  - An if clause with one statement and braces

```
if ($qualified) {
        // This expression is equivalent to $qualified == true.
        echo 'You qualify for enrollment.';
}
```

  - If and else clauses with one statement each and no braces

```
if ($age >= 18)
    echo 'You may vote.';
else
    echo 'You may not vote.';
```

- Why you should always use braces with else clauses

```
if ($age >= 18)
        echo 'You may vote.';
else
        echo 'You may not vote.';
       $may_vote = false;      // This statement isn't a part of the else clause.
```

- Braces make your code easier to modify or enhance

```
if ($score >= 680) {
        echo 'Your loan is approved.';
} else {
        echo 'Your loan is not approved.';
}
```

- How to code if statements with else if clauses
- An if statement with one else if clause

```
if ($age < 18) {
        echo "You're too young for a loan.";
} else if ($score < 680) {
    echo "Your credit score is too low for a loan.";
}
```

- An if statement with an else if and an else clause

```
if ($age < 18) {
  echo "You're too young for a loan.";
} elseif ($score < 60) {
  echo "Your credit score is too fora loon.";
} else {
  echo "You're approved for your loan.";
}
```

More examples

An if statement with two else if clauses and an else clause

```
$rate_is_valid = false;
if(!is_numeric($rate)) {
   echo 'Rate is not a number.';
} else if($rate < 0) {
   echo 'Rate cannot be less than zero.';
} else if($rate > 0.2) {
   echo 'Rate cannot be greater than 20%.';
} else
    $rate_is_valid = true;
}
```

An if statement to determine a student's letter grade

```
if ($average >= 89.5) {
  $grade = 'A';
} else if ($average >= 79.5) {
  $grade = 'B';
} else if ($average >= 69.5) {
  $grade = 'C';
} else if ($average >= 64.5) {
  $grade = 'D';
} else {
  $grade = 'F';
}
```

How to code switch statements

- A **switch** statement is a convenient way to express a certain form of if statement. Specifically, it can be used in place of an if statement with multiple else if clauses in which one expression is tested for equality with several values.
- The switch statement starts by evaluating the switch expression in the parentheses.
- After evaluating the expression, the switch statement transfers control to the **case** label that has the value that matches the value of the expression. Then, it executes the statements for that case. It stops executing when it reaches a **break** statement or the end of the switch statement.
- The **default** case is optional and may be omitted. If included, you can only have one default case. It is usually the last case in the switch statement, but it can be anywhere.
- The values in the case label may be literal values or they may be expressions. If a case doesn't contain a break statement, execution "falls through" to the next label.
- You can nest if statements or other switch statements within the cases of a switch statement.

- A switch statement with a default case

```
switch ($letter_grade) {
        case 'A':
                $message = 'well above average';
                break;
        case 'B':
                $message = 'above average';
                break;
        case 'C':
                $message = 'average';
                break;
        case 'D':
                $message = 'below average';
                break;
        case 'F':
                $message = 'failing';
                break;
        default:
                $message = 'invalid grade'; break;
}
```

- A switch statement with fall through

```
switch ($letter_grade) {
        case 'A':
        case 'B':
                $message = 'Scholarship approved.';
                break;
        case 'C':
                $message = 'Application requires review.';
                break;
        case 'D':
        case 'F':
                $message = 'Scholarship not approved.';
                break;
}
```

How to code the iteration structures

- How to code while loops
- The while statement executes the block of statements within its braces as long as its conditional expression is TRUE.
- When you use a while statement, the condition is tested before the while loop is executed.
- A while loop that finds the average of 100 random numbers

```
$total = 0;
$count = 0;
while ($count < 100) {
        $number = mt_rand(0, 100);      // get a random number from 0 to 100
        $total += $number;
        $count++;
}
$average = $total / $count;
echo 'The average is: ' . $average;
```

- A while loop that counts dice rolls until a six is rolled

```
$rolls = 1;
while (mt_rand(1,6) != 6) {        // get a random number from 1 to 6
  $rolls++;
}
echo 'Number of times to roll a six: ' . $rolls;
```

- Nested while loops that get the average and maximum rolls for a six

```
$total = 0; $count = 0; $max = -INF;
while ($count < 10000) {
        $rolls = 1;
        while (mt_rand(1, 6) != 6) { // get a random number from 1 to 6
                $rolls++;
        } $total += $rolls; $count++; $max = max($rolls, $max);
        $total += $rolls;
        $count++;
        $max = max($rolls, $max);
}
$average = $total / $count;
echo 'Average: ' . $average . ' Max: ' . $max;
```

- How to code do-while loops
- The do-while loop is similar to the while loop except that the conditional expression is tested at the end of the loop. As a result, the code inside the loop is always executed at least once.
- A do-while loop that counts dice rolls until a six is rolled

```
$rolls = 0;
do {
    $rolls++;
} while (mt_rand(1,6) != 6);       // get a random number from 1 to 6
echo 'Number of times to roll a six: ' . $rolls;
```

- A do-while loop to find the max and min of 10 random values

```
$max = -INF; $min = INF; $count = 0;
do {
        $number = mt_rand(0, 100);
        $max = max($max, $number);
        $min = min($min, $number); $count++;
} while ($count < 10);
echo 'Max: ' . $max . ' Min: ' . $min;
```

- How to code for loops
- A **for** loop provides a convenient way to code a loop that requires a counter variable.
- The for statement is useful when you need to increment or decrement a counter that determines how many times the for loop is executed.
- Within the parentheses of a for statement, you code an expression that assigns a starting value to a counter variable, a conditional expression that determines when the loop ends, and an increment expression that indicates how the counter should be incremented or decremented each time through the loop.
- When you use the alternate for statement syntax, you end the statement with the **endfor** keyword.
- A for loop to display even numbers from 2 to 10

```
for ($number = 2; $number <= 10; $number += 2) {
    echo $number . '<br>';
}
```

- A for loop to display all the factors of a number

```
$number = 18;
for ($i = 1; $i < $number; $i++) {
  if ($number % $i == 0) {
echo $i . ' is a factor of ' . $number . '<br>';
  }
}
```