Kevin Chen

12/5/16

PHP Notes

Checking Your Setup

* Running phpinfo() to check the PHP configuration.
  + Save a text file with file extension .php to the server root folder. For XAMPP, this is C:\xampp\htdocs
  + Write the opening PHP tag: <?php
  + Then write phpinfo(); on the same line. You should have <?php phpinfo();
  + There won’t be any script in this file, so you don’t need a closing PHP tag.
  + Then open up the file in a web browser.
  + At the very top is the PHP number. Loaded configuration file: location of the main configuration file: php.ini.
  + Configuration section: Look for Core section. display\_errors: tells you whether error messages are displayed on screen. Look at the value under Local Value. Should be on, or else some errors will result in a completely blank screen.
  + error\_reporting: should be 32767.
  + Date section: Look at default timezone value.
* Changing Configuration Settings in php.ini.
  + Find the location of php.ini by checking the PHP configuration, and open it in a plain text file.
  + Lines that begin with a semicolon are comments.
  + error\_reporting should be E\_ALL. (Letters are uppercase b/c it is a constant.) This results in even minor errors being reported.
  + display\_errors should be On.
  + date.timezone should be set to your timezone. Find the correct value in the online documentation.
  + After changes are made, restart the web server then reload the script that runs phpinfo.
  + Additional .ini files parsed attribute would override lists the files that could override values in the main configuration file.
* Alternative Ways to Change Configuration Settings
  + Use a file called .htaccess (only works if server is running Apache). # at the beginning of a line indicates comments. Each setting must be on a separate line. If the configuration directive sets a value, this is the syntax: php\_value [name of directive] [value]. (Example: php\_value error\_reporting 32767) Apache doesn’t understand PHP constants, so you need to use an actual number. If the directive toggles a setting on or off, the line begins with php\_flag, and use on or off for the value. Save this file in the server’s document root folder as .htaccess. You don’t need to restart the web server to see the changes.
  + .user.ini is another configuration file. (Can be used if server supports .user.ini.) Uses same syntax as php.ini.
  + Change configuration settings at run time (least efficient way of doing this): create a php file. For most settings, you use: ini\_set(‘[configuration\_setting]’, [value]); (Example: ini\_set(‘error\_reporting’, E\_ALL);) If you want to change the timezone setting, use date\_default\_timezone\_set(‘[timezone\_value]’);

PHP Basics

* The Big Picture
  + What it can do: examples include doing date and time calculations, changing content, communicate with a database (e.g. MySQL).
  + When someone visits a website, the browser sends a request to the web server. If the webpage is static, consisting of only HTML/CSS/JavaScript, then the server simply responds by sending the necessary files. However, if you use PHP on your webpages, the server sends the page to the Zend engine that powers PHP for processing before sending the response to the browser. If a database is involved the Zend engine sends the request of the database and processes the results.
  + PHP is designed to be an embedded language. Often embedded into HTML in HTML markup.
  + If you view source code of a page, you don’t see PHP code because PHP is a server-side technology (PHP code remains on the server).
  + Use the filename extension .php, even if pages contain only HTML.
  + PHP tags: opening tag is <?php. Don’t use <? as the opening tag. Closing tag: ?>. This is essential if your PHP script is embedded in HTML, but it can be left out if there is no code following the PHP script.
  + Store pages inside the server root folder (often htdocs, though sometimes www wwwroot, public\_html). Sometimes files can be stored outside the server root for security reasons.
  + Always view pages using the URL. <http://localhost/path/to/file/file_name.php>
* Choosing and naming variables
  + Naming variables: variables always begin with $.
  + Don’t use reserved names: $this and reserved variables (find online doc)
* Displaying text and numbers
  + Don’t need to specify type of variable stored in php. $variable\_name = value;
  + To display the value, use echo value; (Example: echo “Hello World”;) Optionally, you can use parenthesis between the value. print works exactly like echo except echo can display a series of values separated by commas whereas print cannot.
  + Can also use accented letters for variable names.
* Comments and white space
  + Single line comments: begin with two forward slashes (like Java) or begin with # symbol.
  + Multiple line comments: /\* multiple\_lines\_of\_comments \*/ (just like Java)
  + White space: As long as the code is not ambiguous, you can remove white space. (e.g. $my\_var=“Hi”;). You can also add a shit ton of white space.
* Difference between single or double quotes, and escape sequences
  + Can use single quote inside double quote and use double quote within single quote normally. But if you use double quote in double quote or single quote within single quote, you need to use escape sequence of \‘ or \“
  + Double quotes can incorporate variables inside the string. Example: echo “I love $book” would actually display “I love Lord of the Rings” if $book stored “Lord of the Rings”. Single quotes would display the literal string: “I love $book”.
  + Double quotes process special escape sequences: \“, \n, \t, \r (carriage return), \\, \$. The only one that works in single quotes is \‘
  + Most coding standards recommend single quotes.
* Joining Strings
  + Instead of using “+” as the concatenation operator, PHP uses a dot (“.”).
  + Alternatively use double quotes to join two variables. Example; “variable\_1 variable\_2”. This works because double quotes can incorporate variables.
  + In Java, you can use += as a combined concatenation operator. In PHP, it’s .=
  + Browsers don’t display carriage returns and new line characters. But these can be seen by viewing the page source.
  + Shorthand for echo: Instead of writing <?php echo string ?>, you can instead write <?= string ?> (This only works for displaying a single string in the PHP block.)
* Using heredoc syntax to avoid problems with quotes
  + If you have a lot of escaped quotes inside a string, then it can get messy. Use heredox syntax to solve this problem.
  + $variable\_name = <<< identifier

actual\_string

identifier;

* + Identifier indicates where the end of the text is. Can use alphanumeric characters and underscore. Often is all caps.
  + Now all quotes don’t need to be escaped.
  + There cannot be anything after the opening identifier (even a space screws it up).
  + Similarly, nothing else can be on the line with the ending identifier and semicolon (even an additional space screws it up).
  + Useful for building SQL queries.
* Storing multiple values in an indexed array
  + Two types of arrays: indexed and associative. We will focus on indexed.
  + $array\_name = array(value\_1, value\_2, …, value\_n);
  + Can’t automatically convert array to String. Inspect all contents of an array using the function print\_r($array\_name);
  + Use the shorthand in JavaScript to create an array: $array\_name = [values\_separated\_by\_commas];
  + Add elements to an array: $array\_name[] = additional\_element;
  + Access individual elements: Just like Java: $array\_name[index];
  + count(array\_name) returns the size of the array.
* Storing values in an associative array
  + Each array element is identified by a string instead of an index, resulting in the array storing key-value pairs.
  + Syntax is same as indexed array except for the values inside the parenthesis or brackets. For each element, write: key => value, and separate elements by commas. Example:

$descriptions = [

‘Earth’ => ‘mostly harmless’,

‘Marvin’ => ‘the paranoid Android’

];

* + Accessing elements: $array\_name[key];
  + Adding element to the array: $array\_name[key] = value;
  + print\_r works here just like it does in indexed array.
  + Unlike indexed arrays, you can’t put an associative array element directly inside a double-quoted string. Solve this problem by surrounding the associative array element with curly brackets {}
  + When you have only command in a PHP block, you don’t need a semicolon (but you can still have it.)

Making Decisions with Conditional Statements

* The truth according to PHP
  + Explicit Boolean values: true or false. These characters are case-insensitive.
  + Implicit Boolean values (aka truthy and falsey values): These are false: 0, 0.0, ‘0’, ‘’, “”, an empty array, NULL (including unset variables), a SimpleXML object created from empty tags. Everything else it true (even -1).
  + A PHP paradox: $OK = ‘false’; $OK evaluates to true. Only an empty string is false.
* Conditions and Comparisons
  + Syntax for if statement is like Java and it works like Java. elseif is normally written as one word, but you can put a space in between.
  + === is identical, whereas == is equal. !== is not equal, whereas !== is not identical. If you compare two variables with the same value, they are equal. But if they are not the same data type, they are not identical.
  + Note = does not result in an error, but instead always returns true (assuming the assignment succeeds) because a value is an implicit true.
  + Logical operators are just like in Java (&&, ||, and !)
* Alternative syntax for conditional statements
  + Instead of curly braces, use a colon after the condition. Add endif at the very end. Example:

if (condition) :

statements

elseif (condition) :

statements

else :

statements

endif;

* + One important difference: elseif must be written in a single word in this alternative syntax. (endif also must be a single word.)
* Switch Statement
  + Syntax example:

switch($var) {

case value1:

//do this if $var == value1

break;

case value2:

case value3:

//do this if $var == value2 || $var == value3

break;

default:

//do this if nothing else matches

* + Switch statement does not have the equivalent of the “and” operator.
  + Can use comparison operator instead using syntax: case $variable\_name < value (or other comparison operator)
* The Ternary Operator
  + $var = (condition) ? value1 : value2;
  + This is equivalent to:

if (condition) {

$var = value1;

} else {

$var = value2;

}

* + If the condition is a variable and if there is nothing between the question mark and the colon (value1 is not there), then if the condition is evaluated to true, then the variable is set to the condition variable, and if the condition is evaluated to false, then the variable is set to the value to the right of the colon (value2).
  + In the above case, if condition variable is not initialized, then the condition is evaluated to false and a warning is shown.
* Setting a default value
  + Null-coalesce operator (PHP 7): ?? instead of ?:
  + If the variable on the left of the operator is null or doesn’t exist, the operator skips over it without raising an error and assigns the value on the right.
  + ?? Can be chained. The operator goes from left to right until the value is no longer null. Example: $variable = $value\_1 ?? $value\_2 ?? value\_3 ?? 25;
  + isset(variable) checks if the variable exists or not. This can be used instead of null-coalesce operator.
* Additional Notes
  + Remember that PHP is supposed to be embedded inside HTML. The code inside a block (e.g. conditional statement block) does not have to be PHP; it can still be HTML

Doing Calculations with PHP

* Arithmetic Operations
  + +, -, \*, /, % are same between Java and PHP.
  + -$number can be used to negate the number.
  + $a \*\* $b can be used to take the power of a to b. (Min PHP 5.6). If using version before PHP 5.6, use pow(number, exponent);
  + Modulus operator converts the values on either side to integers before performing the calculation.
  + There is no integer division in PHP. It will return a decimal if there is a remainder. If you want to do integer division, do . Alternatively in PHP 7 or above, you can do intdiv(numerator, denominator) instead.
* Reassigning the result of a calculation to the same variable: +=, -=, etc. just like Java.
* Increasing or decreasing a number by one
  + ++, and --. If the operator is after the variable (e.g. $number++), then the value is changed after any other calculation; it does not affect the calculation.
  + If the operator is before the variable, the value is changed first, and the new value becomes part of the calculation.

Using Loops for Repetitive Actions

* while loop, do…while loop, continue, break: just like Java
* for loop: just like Java
* foreach loop
  + Syntax for indexed array:

foreach($array\_name as $value) {

// do something with $value;

}

* + Syntax for associative array:

foreach($array\_name as $key => $value) {

// do something with $key and $value;

}

* Alternative syntax for loops
  + Replace opening curly bracket with a colon, and replace closing curly bracket with endfor, enwhile, or endforeach depending on the type of loop.
  + No alternative syntax for do…while loop.

Using Functions and Objects

* Using PHP’s built-in functions: syntax, return, parameters,
  + phpversion(); displays the PHP version.
  + strtolower($string); makes each letter in $string lower case.
  + Some functions take optional arguments: strip\_tags($string) strips out all tags from the string. But there is an optional second parameter string that tells what type of tags to keep. Example: “<p><a>” will keep all paragraphs and links.
  + sort($array) sorts the array in order and rsort($array) sorts in reverse order. These modify the value of the arguments since they pass by reference.
* Understanding PHP documentation: learn to look at online documentation
  + Square brackets around a parameter indicates optional parameters.
  + If a parameter is assigned to a value in the method declaration, that is the default value the parameter takes if no value is passed for the parameter.
  + If you want to combine two constants, you need to separate them using the bitwise or operator, which is a single vertical pipe “|”.
* Creating custom functions
  + Syntax:

function name($argument1, …, $argumentn) {

//lines of code

}

* + Common practice to prefix custom functions with a combination of three or four letters followed by an underscore.
  + (function\_exists(“function\_name”) checks if a function exists or not.)
  + (abs(number) returns the absolute number of number.)
  + You can declare a function after using it. But if a function is in an external file, the file must be loaded first before you can use the function.
* Passing arguments in value and reference to a function
  + Just like C++, & in front of a parameter in the method declaration indicates passing by reference. Only variables can be passed by reference.
  + (var\_dump(variable) inspects the contents of a variable.)
* Using an anonymous function as a callback
  + (implode(string, array) separates all elements in array with the string)
  + Can sort using a user-defined comparison function with usort(array &$array, callable $value\_compare\_func);
  + Instead of creating a function and then passing it into usort, you can create the function within the parameter of usort as an anonymous function. Same syntax as normal function except function name is removed.
  + (SpaceShip operator (PHP 7): $a <=> $b. If $a < $b, then returns -1, if $a > $b, then returns 1, and if $a = $b, then returns 0. Can create an array if you want to have a “tie-breaker” in case the first two values are the same.)
  + (Arrays can be compared directly. Example [$a, $b] < [$c, $d]. First compares $a and $c normally, but if they are the same, then compares $b and $d, and so on.)
* Understanding PHP Objects and Classes: mostly just like Java
  + Access properties and methods using the arrow operator “->”. Example: car1->milage; car1->accelerate();
* Comparing the date() function with the DateTime class
  + (date() function, strtotime() function, DateTime class, DateTimeZone class – refer to online documentation)

Including External Files

* Using Server-Side Includes
  + Includes the contents of one file inside another
  + Advantages: files are merged by the server: doesn’t result in multiple requests from the browser. External files can contain PHP, HTML, or text. Code reuse. Common page elements updated by editing a single file.
  + Four include commands: include, include\_once, require, require\_once. For include, PHP attempts to continue processing even if the external file is missing, whereas PHP would stop for require. Once prevents the file from being included more than once (important for files with function or class definitions).
  + Using Includes for Web Pages: the HTML should be valid after all files are merged. Use HTML fragments to built a complete page. Page shouldn’t have multiple head and body elements. Start with a static page, then cut and paste.
  + Filename extension: include files can use any filename extensions (as long the parent file is capable of processing PHP). Some developers use .inc. Most servers treat .inc as plain text (so sensitive content can exposed). More secure to use a double filename extension such as .inc.php or simply .php. Include files can be stored outside the server root.
* Using include and require
  + include and require are not functions. They are PHP Language Constructs. Thus using parenthesis is optional.
  + Syntax: include ‘./path/to/file’. (“./” is a Document Relative Path. Can also use Absolute Path. Cannot use a path relative to the Site Route, or “\”)
  + Same syntax for the other include commands. Replace “include” with the appropriate command.
  + Remember, if the function definition is in an external file, that external file must be included first before you can call the function.
* Using links in an include file
  + Relative links in external files won’t normally work because those links are relative to the external file, not the parent file containing those external files.
  + To fix this, all links inside an included file need to be relative to the site root. (URL begins with “/” instead of “./”.) You can replace all document relative paths in the included files with the path to the site route, but a much more elegant solution is to store the URL to the site root in a variable, and echo that variable inside the included file before the rest of the path of the URL for each URL.
* Examining the include\_path
  + Using the server’s include path is another way to tell PHP where to find included files
  + Include path specifies where PHP will look automatically for server-side includes if a path hasn’t been specified in the script.
  + Use the function get\_include\_path() to know what the include path is. Each path is separated by a colon (Mac) or semicolon (Windows), and PHP searches each path in order.
  + Can change the value of the include path in main\_configuration\_file\_php.ini, and can change during runtime using the function set\_include\_path(). An elegant way to add paths to the server’s include path is: set\_include\_path(get\_include\_path() . PATH\_SEPARATOR . ‘[additional\_absolute\_path]’);

Handling Errors

* Dealing with PHP errors: unlike HTML, browsers aren’t able to fix the error easily.
* Why is my page blank or incomplete?
  + Completely blank or incomplete screens can occur if display errors is turned off in the PHP configuration and you got errors.
  + Temporarily display errors at runtime using ini\_set() method (look above for details). Pass in parameters ‘display errors’ and ‘1’
  + But if there is still a blank/incomplete screen, then that means you got a syntax error. Can only be resolved by looking at the code or by using a local testing server with display errors turned on.
* Tracking down common parse errors
  + Errors are sometimes not on the same line indicated by the PHP error message. Often is something before the line indicated.
* Dealing with T\_ENCAPSED\_AND\_WHITESPACE
  + This error means that you’ve embedded an associative array element in a double quoted string. To do this, you must wrap the associate array element in a pair of curly braces.
  + Do not add spaces in between the curly braces and the associate array element, or else you get this error as well.
* What to do with “failed to open stream”
  + A server-side include is missing.
* What does “headers already sent” mean?
  + The function header() can be used to ascend an http header to redirect to another page. It does so by passing in the parameter: ‘Location: path/to/other/file.php’.
  + Any whitespace outside the php tags in a server side includes sends out to the browser preventing the header function from working since the header has already been sent.
  + Two solutions: eliminate whitespace outside of php tags OR delete php closing tag if there is only php in that file.
  + Normally the output\_buffering is set up to four KB of output, often preventing you from getting this error, but if it is set to zero, you would always get this error.
* What does undefined index, offset, or constant mean?
  + Undefined index: key for associative array does not exist.
  + Undefined offset: index for indexed array is outside of the array bounds.
  + Undefined constant: constants in PHP do not have the $ in front of it. Typing the name of a constant that does not exist would result in this error. In an associative array with keys of type string, if you forget to put quotes around the key, PHP would think that the key you put is a constant (and chances are it is undefined). Nevertheless, it still works as PHP assumes you meant to put quotes around the string (though you get a warning), but this is bad practice.
  + Undefined variable: trying to use a variable that is not defined.

Emailing the Contents of an Online Form

* Understanding the difference between POST and GET
  + One of the most useful applications of PHP is processing user input from an online form.
  + What happens to data in an online form depends on whether it’s submitted using the post of get method.
  + By default, forms are submitted using get method, though regardless it is often specified in the method attribute of the form tag.
  + Action attribute tells the browser where to send the data for processing. However, you may also add the processing script in a conditional statement and execute the code only when the form is being submitted. This is known as a self-processing form.
  + Leaving the value of the action attribute empty results in the page reloading. But in HTML 5, there must always be a value. You can type the name of the page the form is in, but it is easier to echo the PHP constant $\_SERVER(‘PHP\_SELF’)
  + Check if $\_GET and $\_POST super global arrays are filled or empty. if ($array\_name) will do so. If they are filled, then the form as been submitted, and you can write a script that uses the data submitted from the form.
  + The get array and the post array use the name attribute use the name attribute of each form element to identify the value that’s being submitted. Name attribute can’t have spaces.
  + When you submit using the get method, all the information is included in the URL as a query string, whereas post method does not.
* Displaying error messages conditionally
  + Self-processing forms can be re-displayed with error messages.
  + Need two arrays: $errors and $missing
  + At the top of the form, add a paragraph with an error message asking the user to fix the errors indicated. Surround this paragraph with a conditional statement checking if there is an element in $errors or $missing
  + In the label before each form element, add an error message asking to enter something in the form element below. Surround this with if ($missing && in\_array(‘input\_name’, $missing)), which checks if $missing is not empty and if the $missing array contains a key by the name of the input name.
  + Repeat above step for $errors.
* Making sure required field aren’t blank
  + We need to add a script check that all required fields are not blank.
  + Begin by adding the following code below the $errors and $missing variables:

if (isset($\_POST['send'])) {

$expected = [‘all’, ‘form’, ‘elements’];

$required = [‘required’, ‘form’, ‘elements’];

require 'path/to/processing/script/processing\_script.php';

}

* + Create the processing script that is required by the code above:

<?php

foreach ($\_POST as $key => $value) {

$value = is\_array($value) ? $value : trim($value);

if (empty($value) && in\_array($key, $required)) {

$missing[] = $key;

$$key = '';

} elseif (in\_array($key, $expected)) {

$$key = $value;

}

}

* + This is what the above code does: loops through each key, value pair inside $\_POST, which stores the form element names inside keys and the user input for each element inside their values. Next, it truncates whitespace from $value if $value is not an array. Next, if $value is empty and its $key is required, then add the $key to the $missing array and create a new variable with the $key as its name and set the variable’s value to the null string. ($$key is a variable variable, which creates a variable dynamically. The name of this new variable is the value of $key.) Otherwise if $key is in the expected array, create a new variable with name $key and set its value to $value.
* Preserving user input when a form is incomplete
  + The user doesn’t want to have to retype all fields if he made a mistake in one of the fields.
  + Add a value attribute to each input element if there are elements in the $errors or $missing arrays. The value of the value attribute is stored in the variable variable. This is the code:

<input type="text" name="input\_name" id="input\_name"

<?php

if ($errors || $missing) {

echo 'value="' . htmlentities($input\_name) . '"';

}

?>

>

* + Instead of adding a value attribute, some types of input elements (e.g. textarea elements) require you to add the original user input in between the opening and closing tags.
  + htmlentities($string\_name) converts any special characters (e.g. double quotes) in $string\_name to their corresponding HTML character entities.
* Blocking suspect phrases
  + A common type of attack is email header injection, which attempts to trick your form into sending html email with copies to a large number of people.
  + In the processing script, add $suspect = false. Next, add $pattern = ‘/Content-type:|Bcc:|Cc:/i’; This is a perl compatible regular expression that looks for content type followed by a colon or Bcc followed by a colon or Cc followed by a colon. It begins with a delimiter “/”, “|” means or, contains a closing delimiter “/”, and a lowercase i that makes it a case insensitive search.
  + Create a function isSuspect to see if any of the values are suspect:

function isSuspect($value, $pattern, &$suspect) {

if (is\_array($value)) {

foreach ($value as $item) {

isSuspect($item, $pattern, $suspect);

}

} else {

if (preg\_match($pattern, $value)) {

$suspect = true;

}

}

}

This code first checks if $value is an array. If it is, it checks if each value of the array is suspect. preg\_match sees if there is a match against a perl compatible regular expression.

* + Then call isSuspect, passing in $\_POST, $pattern, and $suspect in this order.
  + If we found an attack, we want to prevent the rest of the script from running. So surround the rest of the script with if (!$suspect).
  + Finally, we need to display an error message if we found suspect phrases. Add a php block before the form and check if the post array has got any values and if suspect is true. If both are true, add a paragraph element displaying an error message.
* Preparing to send email
  + PHP has a function called mail() which hands an email message to the web server’s mail transport agent.
  + Set up a series of variables to pass into as arguments in mail(). $to = “person’s\_name <recipient\_email>”. Person’s name is optional. Separate additional emails by commas.
  + $subject = “subject\_of\_email”
  + Need body of message as well. (This will be done later.)
  + Mail has some optional parameters as well. Can create an array for $headers. (This will be imploded with new line characters). Example: $headers = “From: [webmaster@example.com](mailto:webmaster@example.com)”; Example: headers[] = “Content-type: text/plain; charsest=utf-8”
  + Additional optional parameter: $authorized. Stops spamming by ensuring that the mail comes from an authorized account.
* Automating the reply address
  + We want to add the user’s email address to the header’s array only if no suspect phrases have been found.
  + Inside the if-statement checking if no suspect phrases are found, add:

//Make sure required fields are filled in and that email isn’t empty.

if (!$missing && !empty($email)) :

//This function validates the email. First parameter, INPUT\_POST,

//says that we’re looking in the $\_POST array, next parameter is

//the name of the element in the post array, and the last parameter

//is how we want to filter it. Returns the requested variable (the email)

//on success, or false if the filter fails.

$validemail = filter\_input(INPUT\_POST, 'email', FILTER\_VALIDATE\_EMAIL);

//If it is a valid email, add $validemail to the headers. But otherwise,

//we need to add something to our errors array.

if ($validemail) {

$headers[] = "Reply-to: $validemail";

} else {

$errors['email'] = true;

}

endif;

//If no errors, create headers separated by new line characters.

if (!$errors && !$missing) :

$headers = implode("\r\n", $headers);

endif;

* + In the parent file, locate the label for email. Below where you check if email is missing, add an elseif checking if $errors[‘email’] is set. If it is set, then add a span element that says invalid email address.
* Building the message body
  + In the processing script, create a variable that keeps track if mail has been sent, and set that to false. Call this variable $mailSent.
  + Create a $message variable and assign it to an empty string. We’ll concatenate values later.
  + In the statement checking if there are no errors and nothing is missing, loop through all elements inside the $expected array. Call the temporary variable of this for-each loop $field.
  + Remember we dynamically created a variable for each of the elements in the post array, so check if there exists a variable that has the name of the value inside the expected array and that the variable is not empty. If there exists such a variable, set a temporary variable to that variable’s value; else, set the temporary variable’s value to ‘Not selected’. Call this temporary variable $val.
  + Some form elements submit their values as an array, so check to see if the $val is an array. If it is, join the elements together into a single string separated by commas using the implode() function.
  + Replace all underscores of $field with a space.
  + Use the combined concatenation operator on $message. Add the values $field, followed by a colon and space, followed by two carriage returns and new lines. You may want to uppercase the first letter of $field by using the function ucfirst(*string*)
  + To make each line of the message no longer than 70 characters, use the wordwrap function, passing it the string and the number 70.
  + Set $mailSent to be true.
  + Inside the form script, add a <pre> HTML element, and inside the element, put a PHP script that displays the email sent. If $\_POST is not empty and the mail has been sent, then echo $message and $headers, using HTML entities for special characters (use the function htmlentities(*string*)). Good idea to label the $message and $headers with some text above it and separate them by some new lines.
* Sending the email
  + Instead of simply setting $mailSent to true, we only want to set it to true if PHP successfully mails it. So set it to the value returned by the mail() function. For this function, pass in $to, $subject, $message, $headers, and $authorized.
  + If the mail was not sent, add to the errors array an element called ‘mailfail’.
  + To redirect the user to a “thanks” page, then go to the end of the if-statement in the form script that checks if $\_POST[‘send’] is set. Check if $mailSent is true and use the header(‘Location: path/to/thank\_you\_page.php’) and then use the exit command to prevent the script from running further on the form page.
  + Use the same error message as the error message from detecting a suspect input by changing the condition. Instead of displaying the message only if $suspect is true (in addition $\_POST having elements), also display this message if $errors[‘mailfail’] is set.
  + You can get rid of the test block at the end of the page.
  + To test this form, upload the files onto your remote server. If your email couldn’t be sent, then your company may require the fifth argument to mail to be set. In most cases, the value of this needs to be a string in the format: ‘-f*your\_email\_address*’ (an email address that is valid on your own domain).
* Troubleshooting mail()
  + PHP only sends the email; it doesn’t deliver it. PHP’s responsibility ends as soon as the email has been handed to the server’s mail transport agent.
  + If the email doesn’t arrive: check spam filter, fifth argument in mail(), etc.
  + If this doesn’t help, edit and test mail\_test.php in 09\_10 of the exercise files. If success, then then there’s a mistake with your form script. Otherwise, contact your hosting company.
  + Make sure email hasn’t been disabled on the server.

Dealing with Multiple-Choice Form Fields

* Introduction
  + Input types: radio button groups (select only one value from a group), check boxes (can be used singly or as a group. When used as a group, multiple values can be selected), select (drop-down) menus, multiple-choice lists (like a drop-down menu allowing zero or more values to be selected)
* Handling radio button groups
  + HTML syntax: Do this for each option in the radio button:

<input type=“radio” name=“*common\_name*” value=“*unique\_value*”, id=“*unique\_id*”>

<label for=“*unique\_id*”>*unique\_value*</label>

* + Add ‘*common\_name*’ to the expected array. If there is no value by default, then you need to check if $\_POST[‘*common\_name*’] is set. If it is not set, then set its value to the null string.
  + If the script has been submitted with errors, you want to retain the information. In each of the input tags of the radio button group, check if $\_POST contains elements and if $­*common\_name* is equal to the value of the input element. If it is, echo checked. (Remember in HTML radio groups, adding a “checked” at the end of the input tag would make it checked by default.)
  + To make the radio group a required group, add ‘*common\_name*’ to the required array. Add an error message before the radio button group that displays if no value was selected. Do so by creating a <legend> element with a <span> element inside it that is displayed if $missing contains elements and if ‘*common\_name*’ is in the $missing array.
  + To make one of the radio group selected by default when the user first loads the page, in the input that you want to be selected by default, check if $\_POST does not have elements OR $*common\_name* is equal to the value of the input. If either of these conditions are met, then echo “checked”.
* Handling single checkboxes
  + HTML syntax: <input type=“checkbox” name=“*input\_name*”, id=“*id*” value=“*value*”>
  + To make this a required element, add the id of the input to both the expected and required arrays.
  + Before running the processing script, check if $\_POST[‘*input\_name*’] is set. If it is not set, set it to the empty string.
  + We need to retain data if the checkbox has been selected but there are other problems with the user input. Inside the input tag, echo “checked” if $\_POST has elements and $*input\_name* == “*value*”.
  + We need to show an error message if the checkbox was not selected. Above the input element, display a message if $missing has elements and “*value*” is in the $missing array. (You can display using a span element.)
* Handling checkbox groups
  + All checkboxes in the same group share the same name attribute.
  + Problem with using a single name: only the last selected value will be added to the post array. Resolve the problem by turning it into an array, which is done by replacing the value *input\_name* with *input\_name*[].
  + If the checkbox group is not a required element, then add ‘*input\_name* just to the expected array. Before running the processing script, check to see if no checkboxes are selected (which occurs if $\_POST[‘*input\_name*’] is not set). If it doesn’t exist, then create it and set it to the empty array.
  + We need to retain data if other parts of the script have errors. Inside each of the input elements, check if $\_POST has elements and if the value of the value attribute of the input element is in the $ *input\_name* array. If so, echo “checked”.
  + If you want to require the user to select a certain number of checkboxes, count the number of checkboxes checked and display an error message if needed. Do this using the knowledge that count($\_POST[‘*input\_name*’]) gets the number of checkboxes selected. If necessary, add an element to the errors array: $errors[‘*input\_name*’] = true;
  + Before the checkbox group, check if $errors[‘*input\_name*’] is set. If so, display an error message using the <span> element.
* Handling select menus
  + HTML syntax:

<select name=“*name*” id=“*id*”>

<option value=“*value1*” selected>*Display\_text*</option>

…

<option value=“*valuen*”>*Display\_text*</option>

</select>

(Place “selected” in the option you want selected by default.)

* + If you don’t have value attributes, the text in between the opening and closing tags is used instead for the form data.
  + To make an option not a valid choice for the user to submit, change the value to an empty string. (Typically you do this on the first selection, which has the value “Make a selection”.)
  + Add *name* to the expected and required arrays if you want this to be required. We need to display an error message if the user selected an invalid choice. Above the <select> tag, check if $missing has any elements and if ‘os’ is in the $missing array. If so, display an error message in a <span> element.
  + Choosing which option to be selected by default: For the option you want selected when the page loads initially, check if $\_POST doesn’t have any elements OR $*name* is equal to the empty string. If so, echo selected. For the other option tags, you want them selected by default if the user had submitted a form with invalid data. Check if $\_POST contains elements and $*name* == $*value*, where *value* is the value of the value attribute of the option. If so, echo selected.
* Handling multiple-choice lists
  + HTML syntax: same as select menus except you have additional attributes of size (which is set to the number of options) and multiple in the select tag.
  + Similar to checkbox groups, the data needs to be submitted as an array, or else only the last selected element will be processed. Do so by adding [] to the end of the value of the name attribute.
  + Add its name to the expected array. Note that if the user hasn’t selected any of the options, then this won’t be in the $\_POST array. So before running the process script, check if $\_POST[‘format’] is not set. If so, set it to an empty array.
  + Redisplaying the selected elements if the form has been submitted with other errors: for each of the options, check if $\_POST has any elements and *value* is in $*name* array, where *value* is the value of the value attribute of the option. If so, echo selected.
  + If you want to require the user to select a certain number of elements, count the number of elements selected by typing count($\_POST[‘*name*’]) right before the processing script. If this is an invalid number of items selected, then add an element to the errors array by typing $errors[‘*name*’] = true.
  + We need to display an error message if an invalid number of elements has been selected. Before the list, check if $errors[‘*name*’] is set. If so, display an error message within a span element.
* Where Next?
  + PHP for Web Designers course
  + PHP with MySQL Essential Training
  + Creating Secure PHP Websites