**PHP Library**

**PHP Tags**

* PHP has opening <?php and closing ?> tags
* The first php tags usually appear at the beginning of the page before any html and setup the variables to be used in the rest of the page (processed before the html loads)
  + Additional short php tags are used within the html section to retrieve stored data (dynamic data within an html page)

**Comments**

* Commenting is the same as JS, with single line // and multiline as /\* comments \*/
* You can use hashtag # to do a single line comment as well

**Basic Syntax**

* Statements end with a semicolon;
* Whitespaces (spaces and returns) are ignored, so you can span statements over multiple lines
* **Variables**
  + Always begin with $ ($var)
  + Are case sensitive
  + Can’t begin with a digit or two underscores
  + Can contain letters, numbers, and underscores
  + Can’t contain special characters
  + Can’t use special variables within php such as $this (used for objects)
  + Use either camelCase or underscores to separate words in a variable (pick one)
* **Constants**
  + Use the define() function to declare constants
  + Constant values cannot be changed
  + Use all caps for constant names (no $ used as with variables)
    - define(‘CUTEST’, ‘Ryan’);
  + Constants can be any data type

**Data Types**

* Integer
  + Positive or negative whole numbers
* Double
  + Positive or negative decimal numbers (aka floats, real numbers)
  + Can use scientific notation (5.65e-9)
* Boolean
  + True/False
* String
  + Text and text-like (use single or double quotes)
* Array
  + Holds multiple values or one or more data types
* Object
  + Holds properties and methods (data and functions)

**Retrieving data from the GET and POST forms**

* **GET array**
  + Use when the page is going to get (read) data from the server
  + Do NOT use when sending sensitive information as key/value pairs are appended to a url for submission to the server and visible
  + Do NOT use if sending more than 4 KB of data
  + Can bookmark the parameters sent
  + If Refresh is clicked: will automatically resubmit data
  + **Using the GET method**
    - <form action=”filename.php” method=”get”>labels/input types here for form</form> in your html file
    - Format of data sent via url filename.php?key1\_name=value1&key2\_name=value2
    - Values are stored in the $\_GET array
    - $var1 = $\_GET[‘key1’]; would assign the value of ‘key1’ to the variable $var1
      * It’s not best practice to directly retrieve values from a superglobal variable (like the GET array) like this example
      * Instead, use filter\_input() function along with FILTER\_VALIDATE\_ argument (below)
      * $investment = filter\_input(INPUT\_GET, ‘investment’, FILTER\_VALIDATE\_FLOAT);
    - You can also use the <a></a> tag to directly attach data to a url, and this will always use the GET method
      * <a href=”filename.php?key1\_name=value1&key2\_name=value2”>display text</a>
* **POST array**
  + Use when the page is going to post (write) data to the server
  + Use when sending sensitive information
  + Use for more than 4 KB of data
  + Cannot bookmark parameters
  + If Refresh is clicked: warning dialogue box about resubmitting info
  + Syntax for retrieving is $\_POST[‘key\_name’]; just like with the GET method
    - Best to use filter\_input() function as with the GET method when using superglobal variables
    - $investment = filter\_input(INPUT\_POST, ‘investment’, FILTER\_VALIDATE\_FLOAT);

**Working with form data**

* Use the htmlspecialchars() function or htmlentities() function before displaying user entered text on a webpage (see built-in functions section)
* Be careful when displaying data the user has input to prevent XSS attacks and to display characters correctly, check built-in functions
* Use a php function to populate a dropdown list (example embedded in html) (think I could include a file)
  + <label>Investment Amount:</label> html label

<select name="investment"> select is a dropdown box

<?php for ($v = 10000; $v <= 50000; $v += 5000) : ?> php code for : is a loop

<option value="<?php echo $v; ?>"> first parameter is starting value <= is ending then

<?php echo $v; ?> above is value this line is viewed last is increment

</option>

<?php endfor; ?> need to end the for : loop

</select><br>

* Get data using Default syntax
  + $var\_name = filter\_input(INPUT\_GET, ‘form\_field\_name’, FILTER\_VALIDATE\_type);
    - Text boxes
    - Password boxes
    - Hidden fields
    - Radio boxes
      * If none checked, you may want to use if statements to provide a value if the field is null
    - Text areas (don’t need to check if they’re null)
      * Probably want to use POST method to send so the url sent to the server isn’t huge
    - Drop down lists (single selection and no size=”” only)
      * If size=”” is 2 or more, must check if field is null
      * If multiple options can be selected, must get an array (see example below)
* Supply a default value for null information using php
  + $variable\_name = filter\_input(INPUT\_POST, ‘form\_field\_name’);

if ($variable\_name == null) {

$variable\_name = ‘unknown’;

}

* Get data from Check boxes (different than default syntax)
  + Checkboxes that all have unique name=”” in the html form (no value=”” specified)
    - $variable = isset($\_POST[‘form\_filed\_name’]); // netbeans might not like not filtering POST array data
    - No need to filter\_input because you’re only checking if the POST array has a value set, not the data it contains
  + Checkbox arrays that have the same name=”” but different value=”” in the html form
    - The name=”” box in the html form should contain [] so they are stored in an array
      * Example name=”toppings[]”
    - Should use filter\_input() because you are working with value data
    - Must use FILTER\_REQUIRE\_ARRAY for the php to get the array correctly
    - Should use FILTER\_SANITIZE\_SPECIAL\_CHARS to convert characters to html entities to guard against XSS attachs
    - Best way to process array data is a foreach loop
    - $toppings = filter\_input(INPUT\_POST, ‘toppings’, FILTER\_SANITIZE\_SPECIAL\_CHARS, FILTER\_REQUIRE\_ARRAY);

if ($toppings !== NULL) {

foreach($toppings as $key => $value) { // arrow operator declares variables & keys

echo $key. ‘ = ‘ . $value . ‘<br>’; // processing (this sample just shows the data)

}

}

* Get data from List boxes (drop-down list with multiple options selectable or size=”” greater than 1)
  + Must check to see if it is null & must require and array similar to checkbox array above
    - $var = filter\_input(INPUT\_POST, ‘form\_field\_name’, FILTER\_SANITIZE\_SPECIAL\_CHARS, FILTER\_REQUIRE\_ARRAY);

if ($var !== NULL) {

foreach ($var as $key => $value) {

// enter processing code here

}

} else {

// enter processing if the array is empty ($var = NULL) here

}

**Using echo and print statements** to send strings back to the browser

* Use echo statements to accept multiple parameters (print can’t)
* Print statements return a value of 1 (use as part of expressions)
* Echo statements
  + Syntax
    - echo $var1 // echo ‘Welcome to PHP and MySQL!’;
    - echo ($var1) // echo(‘Name: ‘ . $name); which also works like echo ‘Name: ‘ . ‘$name’;
    - echo $var1[, $var2 …] // can’t use parentheses when echoing multiple variables
      * echo ‘Cost: $’, $cost;
  + Use the htmlspecialchars() function to prevent XSS attacks when echoing inputted info
  + Must use echo statements to display php in returned html
    - <p>Today’s date: <? php echo date(‘m/d/Y’); ?>.</p>
  + <p>Name: <?php echo htmlspecialchars($full\_name); ?></p>
    - The php echo statement is inserted in the html to be sent back to the browser
  + htmlspecialchars not needed for calculated values, but ok to use
    - <?php echo $calculated\_value; ?>
* Print statements
  + Syntax
    - print $var1
    - print($var1) // parentheses are usually preferred
  + Must use print statements in expressions (see below)
    - ($age >= 18) ? print(‘Can vote.’) : print(‘Cannot vote.’); // need to use print because this expression must return values from the print function, and echo does not do this

**Arrays**

* Arrays contain elements that consist of an index and a value
* Can have either integers (0 indexed) or strings for index values (avoid mixing index types, bad practice)
* Length of an array corresponds to the # of elements
* Creating a number indexed array
  + *$array\_name* = array([value1[, value2…]]); // can create empty arrays
  + Separater values in an array with commas
  + Can add values by setting the index $array\_name[5] = 12;
* **Associative arrays** are arrays that have **strings for index values**
  + Uses key/value pairs
  + Creating an associative array
    - *$array\_name* = array(*[key1 => value1[, key2 => value2]]…*);
    - $name = array(‘first\_name’ => ‘Derek’, ‘last\_name’ => ‘Jeter’, ‘number’ => 2);
    - An indexed array with multiple gaps can function like an associative array
* Getting values/referring to an element *$array\_name*[*index*];
  + Null is returned if you get a value for an index that doesn’t exist
  + If key value is a string, be sure to use ‘’ i.e. $name[‘first\_name’];
  + For strings, use double quotes to interpolate
    - Can use braces (optional) to separate if desired (**echo “First name: {$name[0]}”;**)
      * Just use braces all the time because that will make it easier with associative arrays
    - **Associate arrays are weird (probably easiest to just use the {})**
      * echo “First Name: $name[‘first\_name’]”; // generates a parse error
      * echo “First Name: $name[first\_name]”; // First Name: Derek
      * **echo “First Name: {$name[‘first\_name’]}”; // First Name: Derek**
* Adding/changing values in an array
  + *$array\_*name[] = *$value*; Append (add) values to the end of an array (use empty brackets)
    - If no index is specified for an associative array (string indexes), the index is assigned 0!
      * Be careful no to do that
  + *$array*[*$index*] = *$value*; if you specify the index, it adds/changes the value at that spot
    - If you leave gaps with a number indexed array (i.e. last index is 7 and you add an index of 10) NULL values fill between
* **Functions for arrays**
  + print\_r(*$array\_name*);
    - prints all of the index and values stored in the array
  + array\_keys(*$array\_name*[, *$value*[, *$strict*]]);
    - returns the keys from $array\_name that contain an optional $value
    - a strict value of TRUE enforces === for the $value (default is false)
  + define(); function if the values of an array should never change (defines a constant, PHP 7.0 and later)
    - define(‘*$array\_name’*, array(value1[, value2…]);
  + unset(*$array\_name*[*index*]);
    - Leaves a gap in data that is a Null value at the specified index (use array\_values function to fix)
    - unset($array\_name[0]); // sets index 0 to Null
    - unset($array\_name[‘first\_name’]);
    - unset($array\_name); will set entire array to Null
  + array\_values(*$array\_name*);
    - Returns an array with any null gaps removed and values re-indexed
  + count(*$array*);
    - Returns the number of elements (does not count gaps/Null values)
  + end(*$array*);
    - Moves the cursor for the array to the last element in the array
  + key(*$array*);
    - Returns the index of the array element that the cursor is on
  + isset(*$array*[*index*]); form of isset(*$var*);
    - Returns TRUE if the array element contains a value, false if it does not
  + range(*$lo, $hi*[, *$step*]);
    - Returns an array filled with values from $lo to $hi with an increment of $step (1 default)
  + array\_fill(*$start, $count, $value*);
    - Returns an array filled with $count entries of the value $value starting at index $start
  + array\_pad(*$array, $size, $value*);
    - Returns an array with $value added to the end until it contains $size elements
    - A negative $size will add values to the start of the array
  + array\_merge(*$array1, $array2…*);
    - Returns one array with the elements of two or more arrays
    - String keys in $array2 overwrite keys in $array1
    - Indexed (numeric) keys are appended
  + array\_slice(*$array, $index*[, *$len*[, *$keys*]]);
    - Returns part of an array start from $index that contains $len elements
    - If $len is omitted, default is to return the array to the end
    - If $keys is set to TRUE, the original keys are used, otherwise the slice is reindexed to 0
  + array\_splice(*$array, $index*[, *$len*[, *$new*]]);
    - Modifies $array by replacing its elements with the elements in $new starting at $index and replacing $len elements
    - Works kinda like merge, but can overwrite data if $index & $len include data that already exists in the $array
    - The array is reindexed to 0
    - Does not return an array, just modifies the existing $array (no need to store the result)
  + array\_push(*$array, $value*);
    - Adds $value onto the end of $array
    - Same result as *$array*[] = $*value*
  + array\_unshift(*$array, $value*);
    - Adds $value to the start of $array
  + array\_pop(*$array*);
    - Removes and returns the last value from $array
  + array\_shift(*$array*);
    - Removes and returns the first value in $array

Stacks use push and pop to add/remove last in first out (LIFO)

Queues use push and shift to add to the end and remove the first: first in first out (FIFO)

* + array\_sum(*$array*);
    - Returns the sum of the values in the array
  + array\_product(*$array*);
    - Returns the product of the values in the array
  + in\_array(*$value, $array*[, *$strict*]);
    - Returns TRUE if the $value is found in the $array
    - If $strict is set to TRUE, the comparison only returns TRUE if the datatype matches
  + array\_key\_exists(*$key, $array*);
    - Returns TRUE if the $key is used as a key in $array
  + array\_search(*$value, $array*[, *$strict*]);
    - Returns the key if the $value is in the $array, returns FALSE if it isn’t found
    - $strict must find a datatype match
  + array\_count\_values(*$array*);
    - Returns a new array with all values of $array as keys, and the #times that value appears as the new value (kinda like a histogram table)
  + array\_unique(*$array*[, *$compare*]);
    - Returns $array with duplicate values removed
    - $compare is the same, except SORT\_STRING is the default
  + array\_reverse(*$array*[, *$keys*]);
    - Returns $array with elements in reverse order
    - $keys = FALSE (default) the array is reindexed (TRUE preserves original keys)
  + shuffle($array)
    - Reodrders the elements randomly, and reindexes the array
  + array\_rand(*$array*[, *$count*]);
    - Returns a random key from $array (values are not returned)
    - If $count > 1, it returns an array of $count random keys from $array
  + sort\_array(*$array*[, *$compare*]);
    - Sorts the values in the array in ascending order and reindexes the array
    - **$compare settings**
      * SORT\_REGULAR (default) uses relational operators
      * SORT\_STRING each element is converted to a string before comparison
      * SORT\_NUMERIC each element is converted to a number before comparison
  + rsort(*$array*[, *$compare*]);
    - Sorts values in descending order, then reindexes the array
  + assort(*$array*[, *$compare*]);
    - Sorts values in ascending order and keeps the index values (good for associative arrays)
  + ksort(*$array*[, *$compare*]);
    - Sorts keys in ascending order and keeps the values with the keys (associative)
  + krsort(*$array*[, *$compare*]);
    - Sorts the keys in descending order and keeps the values with the keys (associative)
* For loops with arrays (very useful)
  + Use the count(*$array*) function if you want to do something for every element of an array
    - for ($i; $i < count(*$array*); $i++) { // will execute the code contained for every array element
  + Can use if (isset($array[$i]) { in a for loop to execute code for values skipping any nulls
    - Probably easier to just use a foreach loop, as this doesn’t process any null values
* Foreach loops with arrays (commonly used with associative arrays)
  + Declares a variable to work with each value in the array
  + Can also declare a variable to work with the keys
  + Using a foreach loop to make an unordered list for an html page (can use values or key/value pairs)
    - **Values only example**
    - $tax\_rates = array(‘NC’ => 7.75, ‘CA’ => 8.25, ‘NY’ => 8.875);

echo ‘<ul>’;

foreach ($tax\_rates as $rate) {

echo ‘<li>$rate</li>’;

}

echo ‘</ul>’; // would display bullet list with just the values for the tax rates

* + - **Key/value pair example**
    - $tax\_rates = array(‘NC’ => 7.75, ‘CA’ => 8.25, ‘NY’ => 8.875);

echo ‘<ul>’;

foreach ($tax\_rates as $state => $rate) { // code says that $state is key $rate is value

echo ‘<li>$state ($rate)</li>’; // codes for ‘NC (7.75)’ style as each point

**Objects**

* Executing a method of an object
  + *$obj\_name*->*method()*;
* Creating a Class
  + Classes are used to group a similar type of object
    - Objects that have the same class have access to the same methods/code
  + class *Classname* { … } // class names are usually capitalized by convention
    - All items below this line are coded within the braces { }
  + Create properties (stored in variables) within a class
    - can be public (accessed within/outside a class), protected (can be accessed by subclasses but not outside the class/subclass), or private (only accessed within the class, not subclasses)
    - private *$variable\_name*; // can assign default values to variables using = *value* before the ;
      * Default values must be scalar, array of scalar, number literal, string literal, or Boolean
    - Can list multiple properties of the same type together if no default values (otherwise list on separate lines)
      * private *$size, $total\_length, $sex…*;
  + Create constants
    - const *CONST\_NAME* = *value*; // constant names are CAPS by convention
    - Constants cannot be altered
    - Constants are public and can be accessed outside the class/objects
    - Constants can be reference without using an object from that class
  + Referencing a constant
    - Within a class declaration
      * self::*CONST\_NAME*
    - Ouside a class declaration
      * *ClassName::CONST\_NAME*
  + Create constructors/desructors
    - Constructors are methods automatically executed when a new object is created from a class
    - Constructors should be public
    - Often used to initialize variables
    - public function \_\_construct(*$param1, $param2…*) { // parameters are optional

$this->param1 = $param1; // access parameters using $this and store in

$this->param2 = $param2; // $variables

}

* + - Destructors are code automatically executed when an object is no longer available (unset or deleted or at the end of a script)
      * Can be used to close a database connection
        + $this->dbConnection->close();
      * Though not needed if using PDO objects that do this for you
  + Coding methods
    - Typically code public, protected, or private before the function keyword
      * Otherwise, code is the same for functions
    - Common methods are used to *get* & *set* properties
    - To use properties or other functions set within the class, use $this->*name* to access the property (variable) or function with the given *name*
  + Static properties and methods
    - Belong to the class, not any object (works with the class, not an associated object)
      * Can’t use $this-> to access an object
      * Must use self::*staticName* // to access info on the class within the class declaration
        + Use $ with $staticName for properties
      * Must use *className::staticName* // to access info outside the class declaration
        + Use $ with $staticName for properties
    - By default are public (can specify private or protected)
    - private static *$propertyName* = *value;* // can initialize a value
    - public static function *functionName(arguments) {…}* //
    - A use could be to count the number of objects created from a class
      * private static $objCount = 0; // initialize object count at 0
      * include a self::$objCount++; line within the constructor in increment with every object
      * public static getObjCount() {

return self::$objCount;

} // this function would return a count of the number of objects

* Creating Objects
  + Objects of a particular class can be created/stored in a variable in format
    - *$objectName =* new *ClassName(argument list);*
  + The argument list are the arguments needed for the constructor
  + Objects can be stored in variables, as an item in an array, as a property of an object, or pass directly to a function
* Accessing Object’s Properties/Methods (if they are public)
  + *$objectName->propertyName* = *$variable* // store a property of an object in a variable
  + *$objectName->methodName(argument list)* = *$variable* // call a method and store result in a variable
  + echo *$objectName->methodName(arg list);* // would display result of an object’s method
  + Use object chaining to get nested methods
    - If a method returns an object (*methodOne*), you can string together calls to access more properties/functions
    - *$objectName->methodOne()->methodTwo();*
* Copying/Cloning Objects
  + You can set another variable to the same object
    - *$objName1 = $objName2;* // both $objName1 and $objName2 refer to the same object
  + To duplicate an object in a separate new variable, use the clone keyword
    - *$newObj* = clone *$oldObj*; // These are two distinct objects
    - These are shallow copies, and nested objects are not copied (modifying one could modify the other)
* Comparing Objects
  + Use === to see if variables refer to the same object (returns false on clones even if values are the same)
  + Use == to see if objects are the same class and have the same values for every property
* Inspecting Objects
  + class\_exists(*$className*); // returns TRUE if the class has been defined
  + get\_class(*$objName*); // returns the class name of the specified object as a string
  + is\_a(*$object, $class*); // returns TRUE if the $object is from the specified $class
  + property\_exists(*$object, $property*); // returns TRUE if the $object has the $property
  + method\_exists(*$object, $method*); // returns TRUE if the $object has the $method
  + Using the reflection API (<http://www.php.net/manual/en/book.reflection.php>)
* Inheritence
  + Use *extends* create a subclass/childclass of a superclass/parent class
  + class *SubClass* extends *ParentClass* { … }
  + Supply additional properties, a constructor, and methods to the subclass
  + The subclass constructor overrides the parent constructor
    - To use both, call the parent constructor after the subclass constructor in the class definition
    - parent::­­\_\_contruct(*argument list*); // use parent keyword to call parent methods
  + The subclass has access to all public properties and methods of the parent class
    - The subclass can use private properties if there are public methods to work with them set in the parent class
  + Protected modifiers can be accessed from subclasses, but not from outside of either
    - protected *$property* or protected function *method*
* Abstract Classes and Methods
  + Code *abstract* keyword before *class* or access modifier (public, private, or protected) of a method
  + Cannot create an object from an abstract class
    - Issues fatal error
    - Can use to create concrete subclasses
      * Still need to use parent::\_\_construct(); function as with all subclasses
  + Abstract methods should not include code, just ();
    - Can only be created in an abstract class
  + Must use abstract functions within the subclass (no weird syntax to reference)
    - public function *abstractFunctionName*() { … }
    - Fatal error issued if abstract functions are not used in subclasses
* Final Classes and Methods
  + Code keyword *final* before a class name or the access modifier (public etc.) for a method
  + Final classes cannot be used as parents
    - Fatal error if you try to extend a final class into a subclass
  + Final methods will not be overridden by a method in a subclass (subclasses will use the final version)
* Interfaces
  + Unlike other languages, PHP classes can only inherit from one other class
    - But a class can implement multiple interfaces
  + Creating interfaces
    - interface *interfaceName* { … } // can only define constants and methods within braces
      * const *constantName* = *constantValue*;
      * public function *methodName*(*parameter list*);
        + Interface methods must be public
        + Interface methods cannot contain a block of code
      * Interfaces cannot define properties (classes that use them must do that)
  + Using interfaces
    - Create a class that uses interface(s)
      * class *ClassName* extends *ParentName* implements *interface1, interface2…* { … }
    - Must use any interface methods within this class (or fatal error)

**Working with strings**

* Full list of string functions at <http://www.php.net/manual/en/ref.strings.php>
* Very similar to other languages for empty strings (‘’) and null values (null)
* Use single quotes unless you want **interpolation**
  + Using double quotes for strings makes php check entire string for variables that need to be inserted in the string (interpolation)
  + $first\_name = ‘Ryan’;
  + $full\_name = “$first\_name Tucker”;
    - Second example would code Ryan Tucker and needs double quotes since working with a variable
    - Any variable found within double quotes will output its value as a string
    - Use {} if text interferes
      * $message = “Your order includes $count ${item}s”; to make item name plural
* heredoc (to create multiline strings)
  + $message = <<<MESSAGE // the <<<MESSAGE begins the string (doesn’t have to be MESSAGE)

You can type over multiple lines here.

This also does interpolation of variables for $first\_name

or any other person.

MESSAGE; // to close the string (no characters are allowed after heredoc name in opening or closing)

* nowdoc (to create multiline strings, works like heredoc but without interpolation)
  + <<<’NOWDOCNAME’

NOWDOCNAME;

* Use double and single quotes for special purposes
  + $statement = ‘she said, “Hi.”’;
  + $last\_name = “O’brien”;
* **Concatenating** uses a period
  + $first\_name = ‘Ryan’;
  + $last\_name = ‘Tucker’;
  + $name = ‘Name: ‘ . $first\_name; // $name = ‘Name: Ryan’
  + $full\_name = $first\_name . ‘ ‘ . $last\_name // $full\_name = ‘Ryan Tucker’
  + Adding numbers to a string directly or using a variable will convert the number into part of the string
  + .= works like += and will append values to a string
  + $full\_name .= ‘da da da-da da’; // $full\_name = ‘Ryan Tucker da da da-da da’;
* **Escaping characters**
  + Use \ to escape a \ in single/double quote/or heredoc strings (not necessary in nowdocs)
  + Use \ to escape quotation marks (‘ for ‘ and “ for “)
  + Use \ to escape the following characters in “ & heredoc strings (does not work for ‘ & nowdoc)
    - \$ dollar sign
    - \n new line
      * “This is a \nmultiline string.” // This is a
    - \t tab multiline string.
    - \r carriage return
    - \f form feed
    - \v vertical tab
    - \ooo character with the specified octal value (\o + two digit value
    - \xhh character with the specified hexadecimal value (\x two digit value)

**Special String Functions**

* **htmlspecialchars($string [, $quote\_style[, $charset[, $double\_encode]]])**
  + Converts certain HTML special characters ( &, ‘, “, <, and >) to their HTML entities and returns the resulting string (& becomes &amp)
  + Shouldn’t need to use when displaying values calculated by your php script, but ok to use anyway
  + Parameters (must code all four if you want to change $double\_encode to false)
    - $string (to convert) is required, all others optional
    - $quote\_style
      * ENT\_COMPAT (default) only converts double quotes
      * ENT\_QUOTES converts double and single quotes
      * ENT\_NOQUOTES doesn’t convert either
    - $charset specifies the character set of the string supplied (default is ‘ISO-8859-1’)
    - $double\_encode is Boolean that species whether to double encode character entities (default is TRUE)
      * A double encoded &lt; would return &amp;lt; (use false for 4th parameter to turn off)
      * Usually would want this to be set to false
* **htmlentities($string [, $quotes])**
  + Use to display html entities properly in html code
  + By default, only double quotes are converted to html entities
    - ENT\_NOQUOTES for the second parameter, the function won’t convert single or double quotes to html entities
    - ENT\_QUOTES will convert both types of quotes
* empty($var) functions works with strings (either empty string ‘’ or null values will return true)
  + Useful to check/validate data and generate error messages
* strlen($str)
  + Returns the length of a string (characters)
* substr($str, $i[, $len]) function is useful for breaking up and formatting strings (such as phone numbers)
  + Pass the string, then the starting character (0 indexed), and add a length to include if desired (no length specified, then the rest of the string is returned
  + For phone numbers, could require 10 digits only if form validation
    - $phone = 8435559999
    - $phone1 = substr($phone, 0, 3);
    - $phone2 = substr($phone, 3, 3);
    - $phone3 = substr($phone, 6);
    - $phone\_f = ‘(‘ . $phone1 . ‘) ’ . $phone2 . ‘-‘ . $phone3;
    - echo $phone\_f; would return (843) 555-9999
* String position functions
  + strpos($str1, $str2[, $offset]) function checks $str1 for an occurrence of $str2 and returns a value of the start position (returns FALSE if not found)
    - Case sensitive
    - Because an integer of 0 is converted to false by type coercion, use === to check if a strpos function is false
    - Only returns an integer for the 1st found occurrence (like spaces, would only find the first space from the specified starting point)
  + strrpos is the same as strpos but starts at the end of the string (still counts as if starting at the beginning)
  + stripos($str1, $str2[, $offset]) & strripos functions are NOT case sensitive
  + str\_replace($str1, $new, $str2) will return a new string with all occurrences of $str1 in $str2 replaced with $new
    - Case sensitive
  + str\_ireplace($str1, $new, $str2) is not case sensitive
* Trimming white (blank) spaces strings
  + ltrim($str) will trim white space (blank spaces) from the left side of the string
  + rtrim($str) trims white space from the right side
  + trim($str) trims right and left sides
* String padding (adds characters to either or both sides of a string, can be used to align strings in a column)
  + str\_pad($str, $len[, $pad[, $type]])
    - Adds paddings to string ($str) to make it a specified length ($len) by adding characters ($pad, default is a space) to the right side of a string unless $type is specified (STR\_PAD\_RIGHT, STR\_PAD\_LEFT, STR\_PAD\_BOTH)
* Change the case of strings
  + lcfirst($str) returns a new string with the first character converted to lowercase
  + ucfirst($str) converts the first character to uppercase
  + ucwords($str) converts first letter of every word to uppercase
  + strtolower($str) converts all letters to lowercase
  + strtoupper($str) converts all letters to uppercase
* Order of characters in a string
  + strrev($str) returns a new string in reverse order
  + str\_shuffle($str) returns a new string with characters in random order
  + str\_repeat($str, $i) retuns a new string with $str repeated $i times
* Converting strings to/from arrays
  + explode($sep, $str) returns an array of strings that are separated in $str by $sep (character(s) of any length)
    - If $sep is an empty string, then FALSE is returned
  + implode($sep, $sa) returns a string that joins elements from the $sa array and adds the $sep character(s) between them
    - $sep is required but can be an empty string (nothing added between elements)
    - If $sep is ‘,’ then you can create csv
    - If $sep is ‘\t’ then that’s tab separated values
* Converting strings to and from ASCII values
  + chr($value) returns a string with the character specified by $value
  + ord($str) returns the ASCII integer representing the value of the first character in $str
* Comparing strings
  + strcmp($str1, $str2) case-sensitive comparison of which comes first
    - Capital letters come before lower case letters
    - Negative integer if $str1 is first
    - Positive integer if $str2 is first
    - 0 if they are identical
  + strcasecmp($str1, $str2) is NOT case-sensitive
    - Negative if $str1 is first
    - Positive if $str2 is first
    - 0 if the same
  + strnatcmp($str1, $str2) case-sensitive comparison that uses numbers wisely
    - strcmp and strcasecmp would put img10 before img6
    - strnatcmp puts img6 before img10
  + strnatcasecmp($str1, $str2) NOT case-sensitive version of strnatcmp
* Format numbers and strings
  + sprintf() see built-in functions
* Regular expressions (search for matching patterns in text strings)
  + preg\_match(*$pattern, $string*);
    - $pattern should be a string contained within forward slashes
      * $pattern = ‘/Pattern/’; // case sensitive by default
      * $pattern=’/Pattern/i’; // adding the i makes the search case insensitive
    - Returns 1 if the pattern is found, 0 if it isn’t, and false if there is an error in the pattern
      * If testing for FALSE or 0, must use ===
    - Matching special characters requires an escape \
      * \\ backslash
        + need to code \\\\ to match/search for \
        + \\ sends \ then \\ sends the second \
      * \/ forward slash
      * \t tab
      * \n new line
      * \r carriage return
      * \f form feed
      * \. period
      * \xhh Latin-1 char whose value is the hexadecimal digits hh
    - Matching types of characters
      * . any character except a new line character
      * \w any letter, number, or underscore
      * \W any character NOT a letter, number, or underscore
      * \d any digit
      * \D any character NOT a digit
      * \s any whitespace char (space, tab, new line, carriage ret, form feed, vertical tab)
      * \S any non-whitespace char
    - Create a list of characters to match against a single character
      * Use the character class by putting your list in []
      * preg\_match(‘/MB[TF]/’, $string); // searches for MBT or MBF
      * Use ^ to match any character except those in the list
        + preg\_match(‘/[^aeiou]/’, $string); // searches for any non-vowels
        + ^^ to search for not a ^
      * Use - to specify a range
        + preg\_match(‘/[1-9]/’, $string); // searches for any number 1-9
      * Bracket expressions
        + [[:digit:] ] any digit (same as \d)
        + [[:lower:]] lowercase letters
        + [[:upper:]] uppercase letters
        + [[:letter:]] any letter (case insensitive)
        + [[:alnum:]] any letter (case insensitive) or number
        + [[:word:]] any letter (case insensitive), digit, or underscore (\w)
        + [[:print:]] all printable characters including space
        + [[:graph:]] all printable characters excluding the space
        + [[:punct:]] all printable characters excluding letters and digits
      * Can combine these things
        + [^[:upper:]1-5AEIOU] matches anything except an uppercase letter, a number between 1 and 5, or an uppercase vowel
    - Match string position
      * ^ beginning of a string (\^ to match ^)
      * $ end of a string (\$ to match $)
      * \b beginning or end of a word (can’t be in a bracket)
      * \B anywhere besides the beginning or end of a word
    - Group and match subpatterns
      * (*subpattern*) // creates a numbered subpattern group
      * (?:*subpattern*) // creates an unnumbered subpattern group
      * | // or, matches either the left or right subpatterns
      * \n matches a numbered subpattern group
    - Match a repeating pattern
      * {n} pattern must repeat exactly *n* times
      * {n, } pattern must repeat *n* or more times
      * {n,m} subpattern must repeat from *n* to *m* times
      * ? zero or one of the pervious subpattern
      * + one or more of the previous subpattern
      * \* zero or more of the previous subpattern
        + To match a phone number 555-555-5555, (555) 555-5555 or both

‘/^\d{3}-\d{3}-\d{4}$/’

^ at the beginning \d{3} 3 digits

a dash

\d{3} 3 digits

a dash

\d{4} 4 digits $ at the end

‘/^\(\d{3}\) ?\d{3}-\d{4}$/’

^ at the beginning ‘(‘ then 3 digits then ‘)’

optional space ‘ ?’

3 digits

dash

4 digits at the end

‘/^(\(\d{3}\) ?) | (\d{3}-)\d{3}-\d{4}$/’

^ at the beginning matches either…

‘(‘ then 3 digits then ‘)’ and optional space

or

three digits and a dash

followed by 3 digits, a dash, then four digits at the end

* + - Look Ahead Assertions
      * (?=*assertion*) // an extra condition placed on the code that follows
        + ‘/^(?=.\*[[:digit:]])[[alnum:]]{6}$/’ // 6 alphanumeric chars (at least 1 digit)
      * (?!*assertion*) // an extra negative condition on the code that follows
        + ‘/^(?!3[2-9])[0-3][[:digit:]]$/’ // 0-3 then any digit as long as it’s not 32-39
      * Use to enforce password complexity
        + (?=.\*[[:digit:]]) // must contain at least one digit
        + (?=.\*[[:punct:]]) // must contain at least one punctuation mark
        + (?=.\*[[:upper:]]) // must contain at least one uppercase letter
        + (?=.\*[[:lower:]]) // must contain at least one lowercase letter
        + [[:print:]]{8,} // must contain 8 or more characters
    - $pw\_pattern = ‘/^(?=.\*[[:digit:]])(?=.\*[[:punct:]])(?=.\*[[:upper:]])(?=.\*[[:lower:]])[[:print:]]{8,}$/’
  + Can do multiline expressions by coding an *m* just before the closing ‘ in the pattern
  + preg\_match\_all(*$pattern, $string, $matches*)
    - Will find all times the pattern is found in the string and return a count of how many
    - It also stores the matched values in the *$matches* array
  + preg\_replace(*$pattern, $new, $string*)
    - Returns a string that replaces the $pattern with the $new substring
  + preg\_split(*$pattern, $string*)
    - Returns an array of strings that splits the $string on the specified $pattern
* Data Validation with frequently used regular expressions
  + ‘/^[[:digit:]]{3}-[[:digit:]]{3}-[[:digit:]]{4}$/’ // phone number format 555-555-5555
  + ‘/^[[:digit:]]{4}(-[[:digit:]]{4}){3}$/’ // CC info as 5555-5555-5555-5555
  + ‘/^[[:digit:]]{5}(-[[:digit:]]{4})?$/’ // zip code as 55555 or 55555-5555 (? means 0 or more of previous)
  + ‘/^(0?[1-9]|1[0-2])\/(0?[1-9]|[12][[:digit:]]|3[01])\/[[:digit:]]{4}$/’ // date as mm/dd/yyyy
  + preg\_match(*$pattern, $string*); // returns 1 if there is a match

**Working with numbers**

* Mathematical operators are the same as other codes including ++, --, and %
* is\_numeric($var)
  + Returns true if the variable is a number or a string that can be converted into a number
* max()
  + Returns the maximum value passed as arguments
* min()
  + Works like max
* mt\_rand()
  + Generates a random number in the range specified (arguments are included in the range)
* Type casting
  + ($format) ‘$data’;
  + (int) ‘gallons’; // returns an integer value of 0
  + (float) ‘4.2 gallons’; // returns 4.2
* intval($var)
  + Returns an integer value for the supplied variable
  + intval(‘42’); // returns 42
  + Floats are truncated, not rounded
  + Non numeric text stops the function (2,500 would return 2)
* floatval($var)
  + Returns a float for the supplied variable
  + floatval(‘4.2’); // returns 4.2
* The intdiv() function (php 7 and later) will tell you how many times an integer goes into another
  + $x = 8;
  + $y = 14;
  + $z = intdiv($y, $x); // 1 is stored in z (8 goes into 14 1 time)
* Order of operations
  + ++ and -- go before pemdas
  + Use parentheses when needed
* The +=, -=, \*=, /=, and %= works just like other languages by taking the number stored, perfoming the operation with supplied input, then storing the new variable value
* Integer data type (int) can store max values of +/- 2,147,483,647 (PHP\_INT\_MAX is a constant with this value)
  + Floats can store higher/lower values
* Decimal numbers
  + Normal old base 10 counting numbers
  + Almost always used in PHP coding
* Octal (base 8) numbers
  + PHP interprets numbers that begin with a 0 as octal numbers
* Hexadecimal (base 16) numbers
  + PHP interprets number that begin with 0x or 0X as hexadecimal (upper/lowercase letters always ok)
* Generate random number with a specified # of decimal places (mt\_rand if security isn’t an issue)
  + $number = 0 // initialize number variable

$places = 5 // # decimal places you want

for ($i = 0; $i < $places; $i++) {

$number += mt\_rand(0, 9); // generates a random # 0-9 and adds it to what you have already

$number /= 10; // divides that number by 10 to shift the decimal over

} // example could be 0.50297

* Generate random passwords (this one has one symbol, one uppercase, and the rest lowercase letters)
  + Could use random\_int and if/else if statements to decide whether to add symbol, upper/lowercase based on whether random\_int was even or divisible by 3 or something like that
  + $password\_length = 8; // pick any length

$symbols = ‘~!@#$%^&\*()\_-+=;:.,<>?’; // set string full of symbols to choose from

$symbol\_count = strlen($symbols); // says how many symbols there are in the string $symbols

$index = random\_int(0, $symbol\_count - 1); // choose a secure random position in $symbols

$password = substr($symbols, $index, 1); // adds the symbol to the $password

$password .= chr(random\_int(48, 57)); // adds random # 0-9 using chr() ASCII character values

$password .= chr(random\_int(65, 90)); // adds random uppercase letter using chr() ASCII values

while (strlen($password) < $password\_length) {

$password .= chr(random\_int(97, 122)); // adds lowercase letters to reach set length

} // using chr() ASCII values

$password = str\_shuffle($password); // shuffles the characters

**Mathematical Functions**

* is\_infinite($value) returns TRUE if a value is +/- INF
* is\_finite($value) returns TRUE if a value is finite
* abs($value) returns the absolute value
* ceil($value) rounds up to the nearest whole number
* floor($value) rounds down to the nearest whole number
* round($value[, $accuray])
  + If $accuracy is omitted, it rounds to the nearest whole number
  + A positive value for $accuracy specifies how many decimal points to round to
  + A negative value for $accuracy rounds to 10’s and up (-1 rounds to nearest 10, -2 to nearest 100…)
* max($n1, $n2[, $n3…]) returns the value of the largest number
* min($n1, $n2[, $n3…]) returns the value of the smallest number
* pi() returns the value of pi
* pow($base, $exp) calculates $base$exp (both can be floats)
* sqrt($value) returns the square root
* ~~rand($min, $max)~~ returns a random # between min and max (do not use)
* mt\_rand($min, $max) works like rand, but uses the Mersenne Twister algorithm and is faster and better
* random\_int($min, $max) works like the others, but is cryptographically secure (use when security matters)
* intdiv($num1, $num2) tells the # of times $num1 can be divided by $num2 (intdiv(21, 5); returns 4)
* sprintf() see built-in functions for parameters

**Working with Dates**

* DateTime class (newer, object-oriented, preferred, no Y2K38 issue)
  + Each part of the date and time is stored as a separate 64-bit signed integer (no effective limit)
  + **Creating a DateTime object**
    - $var = new DateTime(); // choose any variable name (no args would store current date/time)
    - Arguments passed will usually be accepted if in one of the traditional date formats
    - Arguments can be passed in strtotime(); function format (absolute and/or relative format)
      * new DateTime(‘2018-03-15 13:30:00’);
      * new DateTime(‘+1 day’); (‘+3 weeks’);
      * new DateTime(‘next Friday +1 week 13:30’);
  + Methods of the DateTime object (using methods, *$obj\_name*->*method()*;
    - **Clone before modifying if you want to preserve original DT object (see below)**
    - format($format)
      * Returns a string with the date/time as specified by the format string (like date())
      * Same codes as date function (see below)
      * Use **to display DT from a DT object**
      * example:
        + echo ‘Birthday is: ‘ . $bday->format(‘M/d/Y \a\t H:I a’);
    - setTime($h, $m, $s) // sets the time
      * $obj\_name->setTime($h, $m, $s);
      * Out of range arguments will rollover to the next higher parameter
      * Use checktime() to validate first if necessary
    - setDate($y, $m, $d) // sets the date ($y should be yyyy format)
      * $obj->setDate(yyyy, m, dd); // double digit month should work
      * Out of range arguments will rollover to the next higher parameter
      * Use checkdate() to validate first if necessary
    - modify($str)
      * $my\_date->modify(‘+3 weeks’);
      * Modifies the date/time based on a format string (like strtotime for relative date/time)
    - getTimestamp() // gets the date & time as a timestamp
    - setTimestamp($ts) // sets the date & time using a timestamp
      * **Use this to convert timestamps to DT objects (must have a DT object first)**
      * $new\_dt\_object = new DateTime();

$new\_dt\_object->setTimestamp($old\_timestamp);

* + - add($interval) // modifies the DT object by adding the time specified by a DateInterval object
      * Need a DateInterval object passed as an argument
      * $time\_span = new DateInterval(‘P3W’); // creates a DateInterval object

$due = new DateTime(); // sets the current DateTime

$due = $due->add($time\_span); // adds the interval to the current DateTime

* + - sub($interval) // modifies the DT object by subtracting the time in a DateInterval object
      * Need a DateInterval object passed as an argument
      * works just like add($interval)
      * Can use to have a constant, you must be born before to calculate i.e. voting age
      * $voting\_age = new DateInterval(‘P18Y’);

$dob = new DateTime();

$dob = $dob->sub($voting\_age);

echo ‘You can vote if you were born on or before ‘ . $dob->format(‘F j,Y’);

* + - diff($date)
      * Determines time between current DT object and specified DT object
      * Returns a DateInterval object (negative if $date is earlier than current DT object)
      * Display amount of time between two dates
        + $now = new DateTime();

$key\_date = new DateTime(‘2001-12 last day of the month midnight’);

$time\_span = $now->diff($key\_date);

echo $time\_span->format(‘%R%y years %m months %d days %H:%I:%S’);

// should display a negative sign since $key\_date has already passed

* + - * + Can use conditionals to check for - value of a DateInterval object
  + **Cloning** (copying info from) a DateTime object
    - You must clone data from a DateTime object to modify it without modifying the original
    - If you simply set a new var = to a DT object var w/o cloning, **modifying one will modify both**!
    - $new\_var = clone $original\_object\_name
* DateInterval Class (object oriented, represents a span of time)
  + Creating a DateInterval object
    - *$var\_name* = new DateInterval(‘P*…*’);
    - Parts of the interval string
      * P (begins the interval code, required at the beginning)
      * nY (number of years ‘P1Y’ for 1 year)
      * nM (number of months ‘P2M’ 2 months or ‘P1Y6M’ 18 months)
      * nW (number of weeks ‘P3W’ 3 weeks or ‘P1M2W’ 1 month 2 weeks)
      * nD (number of dats ‘P4D’ 4 days or ‘P2Y6M4W5D’)
      * T (starts the time portion of the code, optional)
      * nH (number of hours ‘PT3H’ or ‘P4DT6H’)
      * nM (number of minutes ‘PT30M’ or ‘P2M6DT3H15M’)
      * nS (number of seconds ‘PT45S’ or ‘P1Y3M6DT4H25M22S’)
  + Methods of the DateInterval class
    - format($format) // used to display date intervals
      * format method codes (if you use codes not stored in the object, it sets them to 0)
        + %R (sign of the interval +/-, looks like this is assigned utomatically)
        + %y (years)
        + %m (months)
        + %d (days) // uppercase letters add leading 0 for all of these
        + %h (hours)
        + %i (minutes)
        + %s (seconds)
        + echo *$interval\_name*->format(‘%m Months, %d Days);
        + echo *$interval\_name*->format(‘%R%yy %mm %dd); // escape char not needed

example from above +1y 20m 15d

* + - * + echo *$interval\_name*->format(‘%H:%I:%S’); //using uppercase for leading 0
* Using timestamps (older method)
  + Full list of timestamp functions (<http://www.php.net/manual/en/ref.datetime.php>)
  + Timestamps use int to represent the #sec since January 1, 1970 GMT (*Unix Epoch*)
  + Using the 32-bit signed integer allows values from December 13, 1901 - January 19, 2038
  + Timestamps will be vulnerable to the *Y2K38* problem in which the 32-bit signed integer reaches its maximum in 2038, causing it to roll over to the lower limit in 1901
    - **Use DateTime objects for future development to solve this problem**
  + Can use traditional mathematical operators to due timestamp comparisons, since they are treated as integers
  + strtotime($str[, $ts]) **also works with datetime objects (very versatile)**
    - Generates a timestamp for most strings that specify dates
    - Accepts strings with a date, a time, or a date and a time
    - **Needs month/year in yyyy-mm format**
    - Absolute templates (string that contains normal date and/or time information)
      * If no time is supplied, the time is set to midnight (00:00:00)
      * If no month, day, or year is supplied, they are set to the current date/time
      * Returns weekday, month, day, year, and time information
    - Relative templates (string that sets time compared to something)
      * +/- days, hours, minutes, etc. (‘+ 1 hour’)
      * Words
        + Tomorrow (sets to midnight of the next day)
        + ‘Tomorrow 10:15am’ would also set the time
        + ‘Next Sunday’
        + ‘last day of’ (will use current month if no month supplied)
        + ‘first day of next month’
        + ‘third Wednesday of’
        + ‘nov second thu of 8am’ (second Thursday of November at 8 am)
  + date($format[, $ts])
    - Gets the current date and returns it in the specified format (which is a string)
      * Can specify a timestamp $ts to return that date/time rather than current date/time
      * $date = date(‘m/d/Y’); returns 09/03/2018
      * Can use . or - instead of /
      * y will return 18 and Y returns 2018
      * date(‘l, F d, Y’); // returns ‘Saturday, September 29, 2018’
      * date(‘Y-m-d \a\t H:i:s’); // returns 2018-09-29 at 11:53:31
        + You must escape format code characters like ‘a’ and ‘t’
        + Non format code characters are displayed as literals (- and : or non code letters)
    - Codes for the date() function (full list available <http://php.net/manual/en/function.date.php>)
      * **Same codes for the format method of a datetime object**
      * D (three letter day Mon)
      * l (lowercase L for full name of weekday)
      * n (month number with no leading 0)
      * m (month number with a leading 0 for single digit months)
      * M (month three letters)
      * F (month full name)
      * j (day of month with no leading 0)
      * d (day of month with leading 0)
      * t (# days in a given month)
      * Y (four digit year)
      * L (leap year (1) or not (0)) // check leap year function below
        + function is\_leapyear($ts) {

return (date(‘L’, $ts) == ‘1’);

}

* + - * g (hours, 12 hour format no leading 0)
      * h (hours, 24 hour format no leading 0)
      * G (hours, 12 hour format with leading 0)
      * H (hours, 24 hour format with leading 0)
      * I (minutes, with leading 0)
      * s (seconds, with leading 0)
      * a (am/pm lowercase)
      * A (AM/PM uppercase)
      * T (timezone abbr.)
      * U (seconds since Unix epoch
  + time() returns the current date and time as a timestamp
  + mktime([$h[, $m[, $s[, $M[, $D[, $Y]]]]]])
    - Returns a timestamp based on the time and date supplied
    - Any values omitted are set to the value from the current date/time
    - Does not validate (if you set 90 for min, it translates as 1 hr 30 min)
  + checkdate($M, $D, $Y) returns true if a valid date is supplied
    - Can generate a checktime function (does not exist) by supplying hour, min, sec parameters and using conditionals to check for validity, and returning true or false based on the result
    - function checktime($h, $m, $s) {

return $h >= 0 && $h < 24 && $m >= 0 && $m < 60 && $s >= 0 && $s < 60;

} // feels like parentheses would help here, but not in Murach’s example

* + getdate([$ts]) returns an array containing the parts of the specified timestamp (current if omitted)
    - key values for the array returned by getdate() (there could be more, see PHP manual)
      * ‘year’, ‘mon’, ‘month’, ‘mday’, ‘weekday’, ‘wday’, ‘hours’, ‘minutes’, ‘seconds’

**Working with Functions**

* Syntax for creating functions
  + function *name\_of\_function*([parameter\_list]) {

// statement(s) to execute;

}

* + Functions need a name, parameter(s) can be optional and separated by commas if more than one
  + End each statement to execute with a semicolon (SQL queries each get one semicolon at the end)
  + Use return $*variable\_name*; if you are wanting to return a value/data from the function
* To call a function, type function name and enter necessary parameters (store returned value in a variable if necessary) (this example is in Functions to work with databases section below)
  + add\_flounder(BB180310, F, 444, 7);
  + $albigutta = get\_gulf\_flounder(A412); // would return an object if using fetchAll(); to return multiple rows in a get\_gulf\_flounder(); function with one parameter (species\_code) and stores in $albigutta
* Creating echo statements to display html code within a function
  + echo ‘<p>Text to display ‘ . *$function\_variable* . ‘any more text.</p>’; // if no text after a variable ‘</p>’
* Scope
  + Variables created outside a function have global scope
  + Variables created within a function have local scope (undefined outside of the function)
  + Making a global variable available to a function
    - global *$var\_name*; // enter this code prior to using the global variable
    - There is also a superglobal variable that stores an array of the global variables $GLOBAL
      * Access variables by *$var\_name* = $GLOBALS[‘*var\_name*’];
* Set a default value for a parameter (makes it optional)
  + When defining the function, provide a value in the function statement (see example)
    - function *function\_name* (*$required\_variable*, *$optional\_variable = 5*) { … }
    - $required variable is required because no default is provided
    - $optional\_variable does not need to be supplied, if it isn’t the value = 5
    - Required variables must be coded first, with optional/default value params after
    - Only scalar values or an array of scalar values can be used
      * String literals, numerical literals, Boolean, or NULL
* Type declarations (for parameters and for return values)
  + Syntax to set types for parameters and/or returns
    - function *function\_name*(*type\_1 $param\_1, type\_2 $param\_2…*) : *return\_type* { … }
  + Types
    - string
    - int
    - float
    - bool
    - array
    - There are others (see php manual)
  + PHP will try to convert to the specified type if possible if the wrong type is passed
    - Can stop this by coding…
      * declare(strict\_types=1);
    - This will throw an exception and produce a fatal error if the wrong data type is passed
* It is possible to create functions that receive differing lengths of parameters
  + See p. 392 Murach or PHP manual if needed
* **Namespace**
  + Create a namespace in a model file
    - Must be the first line of code
    - namespace *ns\_name*; // first way: all functions in the namespace
    - namespace *ns\_name* { /\* functions in the namespace \*/ } // second way **preferred**
    - namespace *ns1\_name\ns2\_name* { /\* functions in ns2\_name within ns1\_name \*/ } // meh
  + Using namespaces
    - require\_once(‘filename.php’); // still need this code to reference the file
    - *ns\_name\function\_name*(*$param1, …*); // reference the namespace in the function call
* **Closures**
  + Provide an inner function access to an outer function’s local variables
    - In the inner function declaration, type *use ($var\_name)* before the {
* **Passing parameters by value or by reference (when calling functions)**
  + By value
    - Original variable not modified
    - Simply supply code input parameter as normal when defining the function
      * function add\_3($number) { … }
  + By reference
    - Will modify or update the original input variable(s)
    - Not necessarily needed to ‘return’ a value, because original variable is modified
    - Use an & before the $ when defining the function
      * function add\_3(&$number) { … }
    - Example to wrap a text string in an html tag
    - The $text variable will be modified, but the $tag variable will not
      * function wrap\_in\_tag (&$text, $tag) {

$before = ‘<’ . $tag . ‘>’;

$after = ‘</’ . $tag . ‘>’;

$text = $before . $text . $after;

}

$message = ‘Turn this into a paragraph.’;

wrap\_in\_tag ($message, ‘p’); // $message is modified by the function

echo $message; // returns <p>Turn this into a paragraph.</p>

**Built-in Functions \*IMPORTANT\***

* Functions use parentheses, arguments are separated by commas, and arguments in brackets [] are optional
* number\_format($number [, $decimals])
  + Returns a number formatted with a decimal, if a second argument is provided, the function will dound to that many decimal places
    - $nf = number\_format(12345.678, 2); will return 12,345.68
  + First input argument must be a double and second must be an integer
* sprintf($format1, $val1[, $val2…])
  + $format uses codes to set the data type of specified values $val
  + The $format string can include more than one data type code
  + Characters NOT between % and data type code are coded literally (part of the string)
  + a % must be used with the data type code
    - %s formats the value as a string
    - %d formats the value as an integer
    - %f formats the value as a float (can expand scientific notation)
    - %e formats the value using exponential notation (supply expanded notation)
    - %c formats as ASCII value (you supply the # as the value, it spits out the character)
    - %b formats an integer as a binary #
    - %o formats an integer as an octal #
    - %x formats an integer as lowercase hexadecimal #
    - %X formats as uppercase hexadecimal
    - To include % sign in $format, use %%
      * sprintf(‘The book about %s has %d pages.’, ‘PHP’, 800); // PHP is string 800 is integer
      * sprintf(‘%s%%’, 4.5); // returns 4.5% as a string
  + Optional format specifiers (all in format code in this order)
    - % (required)
    - Sign (negative signs are shown by default; sprintf(“%+d”, 42); returns +42
    - Alignment (right aligned by default, code a - to left align, and need padding to do it)
    - Padding (default padding is a space, code ‘ or 0 to specify a diff padding char)
      * sprintf(“%10s”, ‘PHP’); codes a padding of 10 for the string PHP right aligned
      * sprintf(“%-10s”, ‘PHP’); codes a padding of 10 with PHP left aligned
      * sprintf(“%-‘\*10s”, ‘PHP’); codes a padding of 10 for PHP left aligned \* as pad char
    - Width (specify a minimum width using an integer) (can use with numbers)
      * sprintf(“%6d”, 42); codes six char (4 blank) then 42
      * sprintf(“%06d”, 42); codes six char 000042
    - Precision (accuracy) (use . followed by a number)
      * For numbers, this specifies the #decimal places (use f for data type)
      * For strings, this specifies the max characters (truncates the string)
      * sprintf(“%010.2f”, 123.456); returns 0000123.46
  + sprintf to generate random hexadecimal color for HTML/CSS
    - $color = ‘#’;

for ($i = 0; $i < 6; $i++) {

$color .= sprintf(“%x”, mt\_rand(0, 15));

} returns value such as ‘#984e1e’

* date($format[, $ts])
  + Gets the current date and returns it in the specified format (which is a string)
    - Can specify a timestamp $ts to return that date/time rather than current date/time
    - $date = date(‘m/d/Y’); returns 09/03/2018
    - Can use . or - instead of /
    - y will return 18 and Y returns 2018
* isset($var)
  + Returns true if variable is set and isn’t null, else false
* empty($var)
  + Returns true if the variable hasn’t been set, is null, or contains an empty string
* strlen($str)
  + Returns the length of a string (characters)
* substr($str, $i[, $len])
  + Returns a substring of the $str variable at position $i, containing the # of char $len (if no $len, then returns from $i to the end)
  + $i is 0 indexed (1st character is 0)
  + $len is a number specifying how many characters (min 1)
* is\_numeric($var)
  + Returns true if the variable is a number or a string that can be converted into a number
* max()
  + Returns the maximum value passed as arguments
* min()
  + Works like max
* mt\_rand()
  + Generates a random number in the range specified (arguments are included in the range)
* **htmlspecialchars($string [, $quote\_style[, $charset[, $double\_encode]]])**
  + Converts certain HTML special characters ( &, ‘, “, <, and >) to their HTML entities and returns the resulting string (& becomes &amp)
  + Shouldn’t need to use when displaying values calculated by your php script, but ok to use anyway
  + Parameters (must code all four if you want to change $double\_encode to false)
    - $string (to convert) is required, all others optional
    - $quote\_style
      * ENT\_COMPAT (default) only converts double quotes
      * ENT\_QUOTES converts double and single quotes
      * ENT\_NOQUOTES doesn’t convert either
    - $charset specifies the character set of the string supplied (default is ‘ISO-8859-1’)
    - $double\_encode is Boolean that species whether to double encode character entities (default is TRUE)
      * A double encoded &lt; would return &amp;lt; (use false for 4th parameter to turn off)
      * Usually would want this to be set to false
* **htmlentities($string, [, $quotes])**
  + Converts **all** HTML characters that have corresponding HTML entities and returns the resulting string
  + Can specify how single & double quotes are converted
  + Shouldn’t need to use when displaying values calculated by your php script, but ok to use anyway
    - Need to use to display some special characters though
  + $copyright = htmlentities(“\xa9 2018”); // xa9 is hexadecimal for copyright symbol, but the function changes that code to &copy; for the copyright symbol in html
  + By default, only double quotes are converted to html entities
    - ENT\_NOQUOTES for the second parameter, the function won’t convert single or double quotes to html entities
    - ENT\_QUOTES will convert both types of quotes
* **nl2br($string [, $is\_xhtml])**
  + Must be used after an htmlspecialchars or htmlentities function or it will simply display <br> tags
  + Will render hard returns to a view in a browser (use when user inserts hard returns and you want to display those hard returns)
  + $is\_xhtml is default true (would use <br /> tags), set to false to use <br> tags
* **filter\_input($type, $variable\_name[, $filter[, mixed options]]) \*IMPORTANT\***
  + Best practice to use this whenever you get values from **superglobal variables** such as $\_GET or $\_POST arrays
  + Will check that values have been set to prevent “undefined variable” errors
  + Validation of entries (i.e. email address or number)
  + Arguments
    - First arg specifies the type of superglobal variable (i.e. INPUT\_GET or INPUT\_POST or INPUT\_COOKIE)
    - Second arg specifies the variable to access (returns null if the variable has not been set, otherwise it returns the value)
    - Third arg (optional) specifies the constant for the filter to apply
      * Default is FILTER\_DEFAULT (no additional filtering)
      * Can add FILTER\_FLAG\_*TYPE* (to allow for certain options) (<http://php.net/manual/en/filter.filters.flags.php>)
        + Validation filters

FILTER\_VALIDATE\_FLOAT (checks for double value)

FILTER\_VALIDATE\_INT (check for integer value)

FILTER\_VALIDATE\_BOOLEAN

FILTER\_VALIDATE\_DOMAIN (checks for domain name label lengths)

FILTER\_VALIDATE\_EMAIL

FILTER\_VALIDATE\_IP (ip address)

FILTER\_VALIDATE\_MAC (mac address)

FILTER\_VALIDATE\_REGEXP (PERL compatible regular expression)

FILTER\_VALIDATE\_URL

* + - * + Sanitation filters

FILTER\_SANITIZE\_EMAIL (Remove all characters except letters, digits and !#$%&’\*+-=?^\_`{|}~@.[])

FILTER\_SANITIZE\_ENCODED (URL encode string)

FILTER\_SANITIZE\_MAGIC\_QUOTES (apply addslashes() function)

FILTER\_SANITIZE\_NUMBER\_FLOAT (removes all charactes except digits)

FILTER\_SANITIZE\_NUMBER\_INT

FILTER\_SANITIZE\_SPECIAL\_CHARS

FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS

FILTER\_SANITZE\_STRING

FILTER\_SANITIZE\_STRIP

FILTER\_SANITIZE\_URL

FILTER\_UNSAFE\_RAW

FILTER\_CALLBACK

* include(‘filename.php’)
  + Inserts and runs the specified file
    - **Passes control to another page**
    - **This is a forward request with only one round trip to the server (the passing control all occurs at the server)**
    - **Faster than a redirect, so use instead of redirect (header (); function) whenever possible**
  + If the function fails (can’t find the file) this causes a warning that allows the script to continue
* include\_once(‘filename.php’)
  + Same as include but makes sure the file is only included once (takes longer to run than include)
* require(‘filename.php’)
  + Works like include, but the script causes a fatal error that stops it if it fails
* require\_once(‘filename.php’)
  + Will only require a file once
* header(‘Location: filelocation.php’);
  + **Redirects** to another page by returning an HTTP response to the browser that contains a location header, which causes the brower to request the specified URL (two round trips to the server)
  + **Two round trips is slower than include(); function**
  + header and include functions often used within if/else statements
  + If redirecting to a filename that is the same as a parameter (like ‘products’) you can use a variable to specify that
    - header(“Location: .?category\_id=$category\_id”);
      * This header redirect will look in the same directory (.) and bring a parameter called category\_id with a value stored in the $category\_id variable to the new page
      * The above example goes to the default file in the current directory and brings the current category\_id with it
* ***header vs. include*** --> looks like if your function is an action query, you need to use header to pass to the new page, but if you are just doing a SELECT query, you can use the include function to pass to a different page
* Type casting
  + ($format) ‘$data’;
  + (int) ‘gallons’; // returns an integer value of 0
  + (float) ‘4.2 gallons’; // returns 4.2
* intval($var)
  + Returns an integer value for the supplied variable
  + intval(‘42’); // returns 42
  + Floats are truncated, not rounded
  + Non numeric text stops the function (2,500 would return 2)
* floatval($var)
  + Returns a float for the supplied variable
  + floatval(‘4.2’); // returns 4.2
* exit(‘string’)
  + The string is optional and can run as exit; or exit();
  + exit(‘Unable to connect to DB.’); will display the string before the browser exits
* die(‘string’)
  + Works like exit

**Conditional Operators**

* Can get unusual results using == or != due to type conversions (see type coercion rules at the end of this section) (use identity or strict operators to solve, or do your own type conversions before comparisons)
  + null == ‘ ‘ returns true
  + null == false returns true
  + false == ‘0’ returns true
  + null == 0 returns true
  + true == ‘false’ returns true (empty string and ‘0’ are converted to false, but all other strings are converted to true)
  + 3.5 == ‘/t3.5 mi’ returns true (string is converted to a number)
  + INF == ‘INF’ returns false (‘INF’ is converted to 0)
  + 0 == ‘ ‘ returns true (empty string converted to 0)
  + 0 == ‘string’ returns true (non numeric stings converted to 0)
* == is equal to (converts data types i.e. 0 == false; would return true)
* === strict or identical operator (0 === false; returns false because 0 and false are different)
* !=
* !==
* ⬄
  + Returns -1 if left is less than right, 0 if equal, and 1 if left is greater than right (spaceship operator)
* <, >, <=, >= (these use type coercion, so convert to same datatype first if necessary)
* ! (not) && (and) || (or)
  + Above are in order of operations if parentheses aren’t used
  + List variable name twice ($state == ‘NC’ || ‘SC’ is incorrect, use $state == ‘NC’ || $state == ‘SC’)
* PHP Type Coercion Rules (performed automatically)
  + Operand 1 vs. Operand 2
    - NULL vs. string (null is converted to an empty string, then compared)
    - Boolean or NULL vs. not a string (convert both to Boolean then compare)
    - String vs. number (convert string to number then compare)
    - Numeric string vs. numeric string (convert both to numbers then compare)
    - Text string vs. text string (compared as if using strcmp() function)

**If Statements**

* Similar to JS
* Test ranges of values in order to streamline code
* if (condition) {

code to execute;

}

* if (condition 1 || (condition 2 && condition 3) {

code to execute;

} else if (condition 4) { // else if is equivalent to elseif in php

code to execute;

} else {

code to exedute;

}

* No braces are needed if only one statement if (!isset($rate)) $rate = 0.075; // include braces tho
* Nested if statements are similar to JS (see example for determining leap year; leap years are divisible by 4, but if they are divisible by 100, it must also be divisible by 400 to be a leap year)
  + $is\_leap\_year = false;

if ($year % 4 == 0) { // check to see if year/4 returns a remainder, if not, year is divisible by 4

if ($year % 100 == 0) {

if ($year %400 == 0) {

$is\_leap\_year = true; // divisible by 4, 100, & 400

} else {

$is\_leap\_year = false; // divisible by 4, 100, but not 400

} else {

$is\_leap\_year = true; // divisible by 4 but not 100

} else {

$is\_leap\_year = false; // not divisible by 4

}

**Switch Statements (a version of if/else if) (use to test a value for multiple cases)**

* Do not use conditional cases, just to check a value against multiple cases
* Syntax
  + switch ($grade) {

case ‘A’:

$message = ‘Very well done!’;

break;

case ‘B’:

$message = ‘Above average!’;

break;

case ‘F’:

$message = ‘Failing’;

break;

default:

$message = ‘invalid grade’;

break;

}

* Syntax with a “fall through” (not recommended because hard to debug)
  + switch ($grade) {

case ‘A’:

case ‘B’:

$message = ‘Scholarship approved!’;

break;

case ‘C’:

case ‘D’:

$message = ‘Needs review’;

break;

} // because no default case was specified, nothing done if all cases fail

* Very useful for the **Controller file in the MVC pattern**
  + switch ($action) { // using the my\_guitar\_shop1 db as example (several cases omitted for space)

case ‘list\_products’:

$category\_id = filter\_input(INPUT\_GET, ‘category\_id’, FILTER\_VALIDATE\_INT);

if ($category\_id == NULL || $category\_id == FALSE) {

$category\_id = 1;

}

$category\_name = get\_category\_name($category\_id); // function from model file

$categories = get\_categories(); // function from model file

$products = get\_products\_by\_category($category\_id); // function from model file

include(‘product\_list.php’);

break; // switch statements need a break!

case ‘show\_add\_form’:

$categories = get\_categories();

include(‘product\_add.php’);

break;

}

**Conditional and NULL Coalesce Expressions (similar to if, can be more confusing to some people)**

* Syntax (can be hard to read, so may want to stick with if statements)
  + (conditional\_expression) ? value\_if\_true : value\_if\_false;
  + $can\_vote\_message = ($age >= 18) ? ‘Can vote.’ : ‘Cannot vote.’;
  + $greeting = (isset($first\_name)) ? $first\_name : ‘Guest’; // uses Guest as greeting if no first name
    - Same thing using **NULL Coalesce (??)** (concise, but may cause confusion with some)
    - $greeting = $first\_name ?? ‘Guest’;
    - Can use **NULL Coalesce** with multiple variables
      * $greeting = $first\_name ?? $email\_address ?? ‘Guest’; // code uses the first non-null value or finally ‘Guest’ if all are null
  + Selecting singular or plural ending
    - $ending = ($error\_count == 1) ? ‘’ : ‘s’; // adds an s if error count isn’t one
    - $message = ‘Found ‘ . $error\_count . ‘ error’ . $ending . ‘.’;
  + Return a value based on a comparison
    - return ($number > $highest) ? $number : $highest; // returns which ever number is greater
  + Nested
    - $value = ($value > $max) ? $max : (($value < $min) ? $min : $value);
    - returns $max if value is higher than max, returns $min if value is lower than min, returns value if value is between max and min (will use $value if within a certain range, otherwise it uses the max or min)

**While and For loops**

* While statements: code executes over and over again while the condition is true, code stops when condition is false (code never executes if condition is never true, but never stops in an infinite loop if never false)
* Both while and for loops are coded similarly to JS
* While loops might never run if the condition starts false
* For statements probably better if you have to use a counter (included as argument)
* Do while loops always run at least once
* Syntax
* while (condition) { // can check the value of a counter like $counter <= 5

code to execute;

code to execute; // often an increment to a variable like $counter++;

}

* do {

code to execute; // counter could be located here or on the next line to execute, must be before while

} while (condition);

* for (initialize variable; condition; increment) { // ex. for ($counter = 1; $counter <= 5; $counter++)

code to execute; // can initialize or increment with any value i.e. $counter += 2

}

* **Alternate syntax For Loops (useful when generating html code for use in a browser like a dropdown list)**
  + <label>Field Name Displayed on Page:</label>

<select name=”field\_name”>

<?php for ($v = initial\_value; $v <= final\_value; $v++) : ?>

<option value=”<?php echo $v; ?>”> // sets the value of the option in the dropdown

<?php echo $v; ?> // sets the text to display for the dropdown

</option>

<?php endfor; ?> // closes the php for loop

</select><br>

* Example of a nested while statement that calculates the average number of die rolls to get a 6, and also keeps a running tally of the maximum number of rolls to reach a 6
  + $total = 0; // start running total at 0

$count = 0; // start count at 0 (haven’t run any yet)

$max = -INF; // any value will be greater than the initial value of $max

while ($count < 10000) { // runs starting $count = 0 and ends after $count = 9999 iteration

$rolls = 1; // first roll

while (mt\_rand(1, 6) != 6) { // get a random # 1-6, and if it isn’t 6…

$rolls++; // …keep going and add another roll to the count

} // but when you do roll a 6 …

$total += $rolls; // add how many it took to a running total

$count++; // increment the counter

$max = max($rolls, $max); // $max will update if $rolls was higher than the max so far

}

$average = $total / $count; // calculates the average

echo ‘Average: ‘ . $average . ‘ Max: ‘ . $max; // displays the average and the max

* Example of a for statement to calculate an investment value

$investment = $1000;

$interest\_rate = 0.1;

$years = 25;

$future\_value = $investment; // initialize future value equal to investment

for ($i = 1; $i <= $years; $i++) {

$future\_value += $future\_value \* $interest\_rate;

}

* Breaks cause a loop to end (any type)
  + You can nest an if (condition) { break; } into a loop to stop it if something happens
* Continue will cause a loop to go back to the beginning (continue;)
  + When used in a for () loop, the counter will increment first (can use to skip even number etc.)
  + Continue ends that iteration and moves to the next

**Exceptions**

* Create a new exception object
  + new Exception(*$message* [, *$code*]); // possible to store in a variable
    - If $message isn’t a string, it’s converted to one
    - If $code isn’t supplied, it’s set to 0
* Throw an exception
  + throw *$exception;* // though typicaly throw/create in the same statement
* Methods of an exception object
  + getMessage(); Returns the message set in the exception object
  + getCode(); Returns the code set in the exception object
  + getFile(); Returns the file in which the exception object was thrown
  + getLine(); Returns the line number at which the exception object was thrown
  + getTrac(); Returns an array containing a stack trace for the exception object
  + getTraceAsString(); Returns a string containing a stack trace for the exception object
* **Use try/catch statements to handle exceptions**
  + try {…}

catch (*ExceptionClass $exceptionName*) {…}

[catch (*ExceptionClass $exceptionName*) {…}]… // can use multiple catches with multiple exceptions

* + - The Exception class has subclasses like PDOException that are more specific
    - Always code the more specific first, and finally let the Exception class be last to catch the rest
  + Example
    - try {

$db = new PDO($dsn, ‘username’, ‘pa55word’, $options);

} catach (PDOException $e) {

echo ‘PDO Exception: ‘ . $e->getMessage();

} catch (Exception $e) {

echo ‘Exception: ‘ . $e->getMessage();

}

* + Can also catch from the Error class the same way as Exception (replace ‘Exception’ with ‘Error’)
  + Catch **ALL ERRORS and Exceptions**  by using ‘Throwable’ instead of ‘Exception’

**Specifiying Directories**

* Using the include function for the example:
  + include(‘childdir/filename.php’); // will open the child directory childdir
  + include(‘./filename.php’); // uses the current directory
  + include(‘../filename.php’); // goes up one directory
  + include(‘../../filename.php’); // goes up two directories
* Using header function
  + header(‘Location: .’); // the current directory (default file in a directory is used if no file specified)
  + header(‘Location: ..’); // up one directory
  + header(‘Location: ./admin’); // down one directory to an admin folder within the current folder
  + header(‘Location: error.php’); // file in the current directory by specifying the filename (no dot needed)
  + header(‘Location: <http://www.murach.com/>’); // use an absolute URL

**Functions to Work With Databases**

* Recall that global variables are not available to functions unless specified ($db example below)
* Escape special characters and place quotes around variables when passing to SQL database
  + quotes(); function // makes inputs safe to use with queries (not as safe as prepared statements)
    - $new\_varName = $db->quote($old\_varName); // you can get $old\_varName from php code
    - $query = “SELECT \* FROM tableName WHERE columnName = $new\_varName”;
* Example function to add a row to a database
  + function add\_flounder($sample\_code, $sex, $total\_length, $age) { // function name & parameters

global $db; // accesses the global $db variable (should be created by database.php)

$query = ‘INSERT INTO flounder // inserts a new row with parameters listed into table listed

(sampleCode, sex, totalLength, age) // column names in the database table

VALUES

(:sample\_code, :sex, :total\_length, :age)’; // values to insert are function params

$statement = $db->prepare($query); // prepare query

$statement->bindValue(‘:sample\_code’ = $sample\_code);

$statement->bindValue(‘:sex’ = $sex);

$statement->bindValue(‘:total\_length’ = $total\_length);

$statement->bindValue(‘:age’ = $age); // bind all values

$statement->execute(); // execute prepared SQL statement

$statement->closeCursor(); // free cursor for other queries

} // end function

* To call a function, type function name and enter necessary parameters (store returned value in a variable if necessary)
  + add\_flounder(BB180310, F, 444, 7);
  + $albigutta = get\_gulf\_flounder(A412); // would return an object if using fetchAll(); to return multiple rows in a get\_gulf\_flounder(); function with one parameter (species\_code) and stores in $albigutta
* To execute a method of an object: $*objectName*->*methodName*(*argumentList*)

**Connecting to a Database**

* Creating a new object from a class
  + new *className(arguments)*;
  + Creating a database object from the PDO (php data object) class
    - new PDO(*$dsn, $username, $password*);
    - DSN syntax for a MySQL database
      * mysql:host=*host\_address*;dbname=*database\_name*
* Connecting to a MySQL database named ‘my\_guitar\_shop1’ located on the local host
  + See “Model” section below for using a Database Class to connect to a database
  + Three arguments are required by MySQL ($dsn, $username, $password)
  + $dsn = ‘mysql:host=localhost:8080;dbname=my\_guitar\_shop1’; // specifies the db location

$username = ‘mgs\_user’; // assuming this username/password combo exists

$password = ‘pa55word’; // whatever password is assigned

$db = new PDO ($dsn, $username, $password); // creates PDO object

* Handling exceptions (when failing to connect to a database)
  + Use try/catch statements
    - The php interpreter will ‘try’ the first statement, and execute it fully if it succeeds
    - If any part of the try statement fails, then it immediately goes to the catch statement
    - Error messages are usually accessed and displayed in the catch statement
  + Handling PDO exceptions
    - try {

$db = new PDO($dns, $username, $password);

echo ‘<p>Your are connected to the database!</p>’;

} catch (PDOException $e) {

$error\_message = $e->getMessage();

echo “<p>An error occurred while connecting to the database: $error\_message </p>”;

}

* + Handling any type of exception
    - try {

// statements that might throw an exception

echo ‘<p>Whatever you want to say if it works, or skip this line</p>’;

} catch (Exception $e) {

$error\_message = $e->getMessage();

echo “<p>Error message: $error\_message </p>”;

}

**Prepared Statements to get and modify database data**

* Use the prepare(*$sql\_statement*) method to return a PDOStatement object (use : to add parameters)
* Use the bindValue(*$param, $value*) method to bind the value to the specified parameter
* Use the execute() method to execute the prepared statement
* Executing an SQL statement that doesn’t have parameters
  + $query = “SELECT \* FROM table\_name”;

$statement = $db->prepare($query);

$statement->execute();

* Executing an SQL statement that has parameters (my\_guitar\_shop1 db as example)
  + $query = “SELECT \* FROM products

WHERE categoryID = :category\_id”; // you supply the requested category ID

$statement = $db->prepare($query);

$statement->bindValue(‘:category\_id’, $category\_id);

$statement->execute();

* fetch() returns an array for the next row in the result set (after a SELECT query) (usually used when only one row is returned from the SELECT statement)
  + If no array is available, the method returns FALSE
* closeCursor() closes the cursor and frees the connection to the server for other SQL statements
* Working with the first row (or only row returned) from a SELECT statement (my\_guitar\_shop1 as example)
  + fetch() returns just the next row (only one row)
  + $query = ‘SELECT productCode, productName, listPrice

FROM products

WHERE productID = :product\_id’;

$statement = $db->prepare($query); // returns a PDOStatement Object for the prepared statement

$statement->bindValue(‘:product\_id’, $product\_id); // binds a variable to the supplied value

$statement->execute(); // stores the result set in a PDOStatement object

$product = $statement->fetch(); // gets the first (and only) array and stores it in the $product variable

$statement->closeCursor(); // closes the cursor and frees the connection to the server

* + The $product variable is an array with productCode, productName, and listPrice keys (created by code above)
* Working with all the rows of a result set from a SELECT statement
  + fetchAll() returns an array for all of the rows in the result set (my\_guitar\_shop1)
    - $query = ‘SELECT productCode, productName, listPrice

FROM products

WHERE categoryID = *:category\_id*;’ // you supply category ID

$statement = $db->prepare($query);

$statement->bindValue(“*:category\_id*”, $category\_id);

$statement->execute();

$products = $statement->fetchAll();

$statement->closeCursor();

* + foreach statement embedded in html will get data for all rows one row at a time (my\_guitar\_shop1)
    - <?php foreach ($products as $product) { ?>

<tr> // table row

<td><?php echo $product[‘productCode’]; ?></td> // no htmlspecialchars() needed

<td><?php echo $product[‘productName’]; ?></td> // because not user inputs

<td><?php echo $product[‘listPrice’]; ?></td> // the data come from the database

</tr>

<?php } ?>

* + foreach statement without use of {} (preferred by many when embedded in html) (my\_guitar\_shop1)
    - <?php foreach ($products as $product) : ?>

<tr> // table row

<td><?php echo $product[‘productCode’]; ?></td> // see why no htmlspecialchars()

<td><?php echo $product[‘productName’]; ?></td> // needed here in example above

<td><?php echo $product[‘listPrice’]; ?></td>

</tr>

<?php endforeach; ?> // easier than when multiple controls are included with lots of }

* + - This method of foreach () : and endforeach; works with if, else if, else, while, and for statements
* INSERT statements (inserting a row into the my\_guitar\_shop1 db)
  + $category\_id = 1; // set a category id to add

$code = ‘strat’; // set a code to add

$name = ‘Fender Stratocaster’; // set a name to add

$price = 699.99; // set a price to add

$query = “INSERT INTO products // insert action query

(categoryID, productCode, productName, listPrice) // specify columns to add

VALUES

(:category\_id, :code, :name, :price)”; // values to add

$statement = $db->prepare($query);

$statement->bindValue(‘:category\_id’, $category\_id);

$statement->bindValue(‘:code’, $code);

$statement->bindValue(‘:name’, $name);

$statement->bindValue(‘:price’, $price);

$statement->execute();

$statement->closeCursor();

* UPDATE statements (updating a row in the my\_guitar\_shop1 db)
  + $product\_id = 4; // specifies (row) item to update

$price = 599.99; // field to update

$query = “UPDATE products // update query to update listPrice for

SET listPrice = :price

WHERE productID = :product\_id”; // specifies to update the record for productID = 4

$statement = $db->prepare($query);

$statement->bindValue(‘:price’, $price);

$statement->bindValue(‘:product\_id’, $product\_id);

$statement->execute();

$statement->closeCursor();

* DELETE statements (deleting a row in the my\_guitar\_shop1 db)
  + $product\_id = 4; // specifies (row) item to delete

$query = “DELETE FROM products

WHERE productID = :product\_id”;

$statement = $db->prepare($query);

$statement->bindValue(‘:product\_id’, $product\_id);

$statement->execute();

$statement->closeCursor();

**STRUCTURE OF PHP APPLICATIONS**

* Use the Model View Controller (MVC) pattern to build complex apps
  + Folders (all within a main folder)
    - Model
    - View
    - Controller
* Model
  + php files that represent the data of the application
  + database.php file should go here
    - database.php file connects to the database using dsn, username, and password, with try/catch blocks to generate error messages, which sent via include() to the database\_error.php file
    - Entire code for database.php file below (populate with dsn, username, and password)
      * Use code below to access the database connection in other functions/model files
        + $db = Database::getDB(); // static syntax to ref. Database class getDB method
    - <?php

class Database {

private static $dsn = ‘mysql:host=*hostlocation*;dbname=*data\_base\_name*’;

private static $username = ‘*username*’;

private static $password = ‘*pa55word*’; // dns location & login info

private static $db;

private function \_\_construct() {} // empty/private constructor prevents objects

from being created from the database class

public static function getDB() { // function returns the $db if its connection is set

if (!isset(self::$db)) { otherwise it creates a new connection

try {

self::$db = new PDO(self::$dsn,

self::$username,

self::$password);

} catch (PDOException $e) {

$error\_message = $e->getMessage();

include(‘../errors/database\_error.php’); //path/filename

exit();

}

}

return self::$db; // this code will only allow one connection/browser

} which is what you want

}

?>

* + Model php pages typically only contain php code dealing with data and no html
  + Model pages frequently consist only of functions that interact with the database
    - SELECT queries and action queries
  + Model pages typically are separated by big ideas, and contain all functions/queries related to that big idea (SELECT & action queries)
    - flounder\_main.php
      * function add\_flounder\_data();
      * function delete\_row();
      * function all\_albigutta();
      * function f\_albigutta();
    - samplers.php
      * all sampler functions
    - maturity.php
      * all maturity functions
* View
  + About View
    - View pages consist of html and php files that represent the user interface (web pages)
    - These pages let you interact with the application
    - These pages contain html tags for the application with some embedded php that display the dynamic data from the database
  + Passing the action variable to the controller via a link button
    - When adding links to a view page, they should bring the action variable to the controller
    - <a href=”.?action=*action\_name*”>*Link Text*</a>
      * If using different folders, make sure to reference correct controller file
      * The ‘.’ only works here because index.php is the default file in the same directory as the current view file
    - The . references the default file in the same directory (in this case index.php)
    - The ?action=*action\_name* sets $action = ‘*action\_name*’ at the end of the url ($GET array)
    - Instead of links, you could use a form button with hidden fields to pass the action variable
  + **DRY (don’t repeat yourself) -> use short files to code things such as headers, footers, nav, aside, and section that are repeated on multiple views**
    - Use <?php include ‘../view/header.php’; ?> (or similar link) before the <main> html tag for a header and <? php include ‘../view/footer.php’; ?> after the </main> html tag for the footer in every file in the view directory
    - These files create consistency throughout the app, and make updating this section for all pages very simple
    - Files should be php type, but only include the html tags with php code you need (leave the <nav> tag in the main view files, and your nav.php file starts with <ul> tag and ends with </ul> tag) (there are no <?php ?> tags to open and close this type of file) (nav example below)
      * <ul>

<!-- display links for all categories -->

<?php foreach($categories as $category) : ?>

<li>

<a href="?category\_id=<?php echo $category['categoryID']; ?>">

<?php echo $category['categoryName']; ?>

</a>

</li>

<?php endforeach; ?>

</ul>

* + - Dropdown list // need to test this as running in a separate file
      * <select name=”category\_name”>

<!-- displays database fields for the dropdown list -->

<?php foreach($categories as $category) : ?>

<option value=”<?php echo $category[‘categoryName’]; ?>”>

<?php echo $category[‘categoryName’]; ?>

</option>

<?php endforeach; ?>

</select><br>

* + - A header file might just contain css link, the app title, and a border, but could also include site navigation links
      * If the header links to the css file, then any view that references the header file will link to that css sheet!
    - A footer file might just contain copyright/date info with a border
  + Various error pages are included here, including a general problem connecting to database\_error
  + The database\_error.php is in this section
    - database\_error.php file is used to display error messages
      * Entire code for database\_error.php file below
      * <!DOCTYPE html>

<html>

<!-- the head section -->

<head>

<title>Page Name</title>

<link rel=”stylesheet” type=”text/css” href=”filename.css”>

</head>

<!-- the body section -->

<body>

<main>

<h1>Database Error</h1>

<p>There was an error connecting to the database.</p>

<p>Any other messages.</p>

<p>Error message: <?echo php $error\_message; ?></p>

</main>

</body>

</html>

* + Display editable data in a table (will need a submit button)
    - You can use the following code to display values in an editable text box that will allow users to update data
    - This code is from a shopping cart application to update quantity
    - The *form\_field\_name* is an array sent to the POST array when the form is submitted
      * *form\_field\_name[$array\_key]*
      * value set to *[$array\_key\_value][‘value’]*
      * For example, the code can send newqty (new quantity information)
      * name=”newqty[<?php echo $key; ?>]”

value=”<?php echo $item[‘qty’]; ?>”

* + - * This sends newqty[$key][‘value’] to the controller page to use with update\_qty() function from a model file with $key set to whatever the $item is
    - To display data, the php code accesses a nested array
      * This displays the current data, but also allows it to be modified
      * The data are stored in *$array\_key\_value[‘value’]*
      * The data can be modified by submitting *form\_field\_name[$array\_key\_value][‘value’]*
      * This sends key/value pair to be modified
    - <td class = “*css\_class*”>

<input type=”text” class=”*css\_class2*”

name=”*form\_field\_name*[<?php echo *$array\_key*; ?>]” lists the key

value=”<?php echo *$array\_key\_value*[‘*value’*]; ?>”> sends keys/values

</td>

* Controller
  + See section on switch statements for coding suggestions
  + The controller section consists of php files that receive HTTP requests from browsers, get the appropriate data from the model files, and return the appropriate view files
  + index.php is commonly in the controller section (could be the only file here)
    - Starts with require\_once(); function(s) to connect to all files in the model including the database.php file and all model files
    - This sets up the $db variable as well as linking (controlling) the necessary data
  + Common to start with an $action variable after the require(); functions, that checks the POST and GET arrays to define the action variable but sets $action to some default value if POST/GET are undefined (first time to that page)
    - $action = filter\_input(INPUT\_POST, ‘action’);

if ($action == null) {

$action = filter\_input(INPUT\_GET, ‘action’);

if ($action == null) {

$action = ‘default\_action’:

}

}

* + Once the action variable is set, the controller has if/else if/else if/else or switch statements that either forward to (include) or redirect to (header) other pages
  + Recommend using switch statements vs. if/else if
    - These switch case (if/else if) statements include the functions from the Model files
    - Data validation occurs here with redirects to various error files as needed
    - Each case (if/else if) statement ends in either a forward (include) or redirect (header)
      * Use include(); if a SELECT query is used, use header(); if an action query is executed
      * The Location: for the header functions can append a variable to the URL to carry through to the new file

**Using Variables for Dynamic Data (filenames, displays, directories, etc.)**

* Store values in a variable, then use variables in the path
  + $image\_filename = ‘../images/’ . $code . ‘.png’;
  + Code above concatenates ../images/ with the code variable then with .png
  + Also useful for something like an $image\_alt value

**Testing and Debugging php Applications**

* Phases of debugging
  + Phase 1: Check User Interface
    - Appearance
    - Tab and enter keys, etc.
    - Radio boxes, drop-down lists, nav components, etc.
  + Phase 2: Check with Valid Data
    - Should eventually test all of the data limits of the app
  + Phase 3: Try to make it fail by testing invalid data and wrong key/mouse presses
* Types of errors
  + Syntax
    - php interpreter is unable to execute code because of rule violations
  + Runtime errors
    - php interpreter displays an error that may or may not stop execution of the script
  + Logic errors
    - Calculation mistakes or incorrect handling of data
* **Tracing execution of code**
  + If running a function, for loop, or calculating multiple variables, use echo statements to include these values in the view of a page
    - This can show you where the error occurs and what that error is
    - Use echo statements to do this, then remove them once the app is debugged
  + Works well with relatively simple applications (tedious with complex ones)
* **Step through code**
  + Click on the #for the line of code in Netbeans to add a “breakpoint” (red square) at each location where you want to pause the script
  + Use toolbar buttons or the keyboard to step through code, checking the value of variables in the debugger window along the way
    - Ctrl+F5 -> Start the debugger
    - F5 -> continue (runs to the next breakpoint)
    - F7 -> Step Into (step through code one statement at a time)
    - F8 -> Step Over (same as into, but doesn’t step through functions)
    - Ctrl+F7 -> Step Out (step out of a function you have already stepped into)
    - Shift+F5 -> Stop the debugger
* **Stack trace**
  + Xdebug installed with Netbeans allows you to click the “Call Stack” tab to view functions in the reverse order in which they were called (useful when many many functions are called in a file)

**Cookies**

* setcookie(*$name, $value, $expire, $path, $domain, $secure, $httponly*)
  + $name is the name of the cookie
  + $value is the value of the cookie (default is empty string)
  + $expire sets the expiration date of the cookie as a timestamp (default is 0, which means it expires when the browser window is closed)
    - Session cookies are set to 0 and expire on closing of browser
    - Persistent cookies expire sometime in the future
  + $path should use ‘/’ so the cookie is available to all directories on the current server, default is the directory of the PHP file that’s setting the cookie
    - Should pretty much always be set to root of the website (so set $path = ‘/’)
  + $domain default is the name of the server that’s setting the cookie
  + $secure if TRUE, the cookie is available only if using HTTPS, default is false
  + $httponly if true, the cookie is only made available through http protocol (not through client-side scripting languages like JS, default is false)
* Getting cookies
  + $\_COOKIE is a superglobal variable, so use filter\_input()
* **Deleting a cookie for a session**
  + Use setcookie() function
    - $name = session\_name();
    - $expire = strtotime(‘-1 year’); // sets expire to a past date
    - $params = session\_get\_cookie\_params(); // get session params
    - $path = $params[‘path’];
    - $domain = $params[‘domain’];
    - $secure = $params[‘secure’];
    - $httponly = $params[‘httponly’];
    - setcookie($name, ‘’, $expire, $path, $domain, $secure, $httponly);
    - (All arguments must be the same as they were in the original cookie)

**Sessions**

* To use session tracking, start a new session, create a session id that is used until the browser is closed
* session\_set\_cookie\_params(*$lifetime, $path, $domain, $secure, $httponly*)
  + $lifetime of session cookie in seconds (default is 0)
  + $path (default is current directory, use ‘/’ to allow all directories)
  + $domain
  + $secure (default false)
  + $httponly (default false)
* session\_start() // **need to set session\_set\_cookie\_params() prior to calling session\_start()**
  + Starts a new session or resumes a previous session if a session ID isn’t found
    - Returns TRUE if successful and FALSE otherwise
    - Function must be called prior to any html output
* Session variable are stored in the global $\_SESSION array
  + $\_SESSION data is stored on the server and is unavailable for manipulation by the user
    - Not necessary to filter\_input to prevent attacks
  + Don’t store objects in the $\_SESSION array for various reasons
* Other session functions (often not needed)
  + session\_name()
    - Gets the name of the session cookie (default PHPSESSID)
  + session\_id([*$id*])
    - If the parameter isn’t specified, gets the current session ID
    - If no session exits, gets an empty string
    - If parameter is specified, sets session ID to that value
  + session\_regenerate\_id()
    - Creates a new session ID for the current session
    - Returns TRUE if successful/false otherwise
    - Can be used to prevent session hacking
  + session\_write\_close()
    - Ends the current session and saves session data
    - Only needed in special cases like redirects
    - Call before using the header() function followed by the exit() function
    - Call before using the session\_destroy() function
  + session\_destory()
    - Ends a session
    - Returns TRUE if successful, false otherwise
    - Want to call if a user logs out of an application
    - Steps
      * Set session array to an empty array ($\_SESSION = array();)
      * session\_destroy();
      * Delete session cookie from browser
        + Use setcookie function
        + See below
    - Must call session\_destroy() after session\_start()
    - Don’t use session\_destory() until after session\_write\_close() (used to save data)
* **Deleting a cookie for a session**
  + Use setcookie() function
    - $name = session\_name();
    - $expire = strtotime(‘-1 year’); // sets expire to a past date
    - $params = session\_get\_cookie\_params(); // get session params
    - $path = $params[‘path’];
    - $domain = $params[‘domain’];
    - $secure = $params[‘secure’];
    - $httponly = $params[‘httponly’];
    - setcookie($name, ‘’, $expire, $path, $domain, $secure, $httponly);
    - (All arguments must be the same as they were in the original cookie)

**Secure Connections**

* See p. 687 Murach for more details
* Redirecting to a secure connection
  + If a user requests http, this will redirect to https
  + include this file (util/secure\_conn.php)
    - <?php

if (!https) {

$host = filter\_input(INPUT\_SERVER, ‘HTTP\_HOST’);

$uri = filter\_input(INPUT\_SERVER, ‘REQUEST\_URI’);

$url = ‘https://’ . $host . $uri;

header(“Location: “ . $uri);

exit();

}

?>

**User Authentication**

* Form-Based Authentication
  + Most common for production websites
  + Gives much control (you design the form/page)
  + Most work to create
  + Sends info to server as plain text (need SSL to encrypt)
* Basic Authentication
  + The browser displays the dialog box asking for username and password
  + Browser determines appearance of the form
  + Not common for production websites
  + Sends info to server as plain text (need SSL to encrypt)
* Digest Authentication
  + Works like basic, but the data is encrypted prior to sending to the server
  + Less secure than Basic with SSL
  + Least common type
* Hashing a password to encrypt it
  + Use password\_hash(); function
  + $password = ‘pa$$word’;

$hash = password\_hash($password, PASSWORD\_DEFAULT);

// can set PASSWORD\_BCRYPT to always use the bcrypt algorithm (right now it’s the default, that may change)

* Setup for implementing Form Based Authentication
  + Create a table for storing usernames and passwords in the database
  + Suggested fields
    - adminID (autonumber) (primary key)
    - emailAddress
    - password (255 chars is best practice)
    - firstName
    - lastName
  + Create admin\_db.php (model) file
    - <?php

// function to add users to the database

function add\_admin($email, $password, $level) {

global $db;

$hash = password\_hash($password, PASSWORD\_DEFAULT);

$query = ‘INSERT INTO *passwordTableName* (emailAddress, password, level)

VALUES (:email, :password, :level)’;

$statement = $db->prepare($query);

$statement->bindValue(‘:email’, $email);

$statement->bindValue(‘:password’, $hash);

$statement->bindValue(‘:level’, $level); // set user level (admin or user)

$statement->execute();

$statement->closeCursor();

}

// function to verify email and passwords against the database

function is\_valid\_admin\_login($email, $password) {

global $db;

$query = ‘SELECT password FROM *passwordTableName*

WHERE emailAddress = :email’;

$statement = $db->prepare($query);

$statement->bindValue(‘:email’, $email);

$statement->execute();

$row = $statement->fetch();

$statement->closeCursor();

$hash = $row[‘password’];

return password\_verify($password, $hash);

}

?>

* + Controller section for protected pages
    - <?php

// start session management and include necessary functions

session\_start();

require\_once(‘model/database.php’);

require\_once(‘model/admin\_db.php’);

// get the action to perform

$action = filter\_input(INPUT\_POST, ‘action’);

if ($action == NULL) {

$action = filter\_input(INPUT\_GET, ‘action’);

if ($action == NULL) {

$action = ‘*default\_action*’;

}

}

// If the user isn’t logged in, force the user to login

if (!isset($\_SESSION[‘is\_valid\_admin’])) { // could use diff than ‘is\_valid\_admin’

$action = ‘login’;

}

// Perform the specified action

switch($action) {

case ‘login’:

$email = filter\_input(INPUT\_POST, ‘email’);

$password = filter\_input(INPUT\_POST, ‘password’);

if (is\_valid\_admin\_login($email, $password)) {

$\_SESSION[‘is\_valid\_admin’] = true; // again with variable name

include(‘view/*mainViewPage*.php’);

} else {

$login\_message = ‘You must login to view this page.’;

include(‘view/login.php’);

}

break;

\\ other cases here that require you to be logged in

case ‘logout’:

$\_SESSION = array(); // clear all session data

session\_destroy(); // clean up the session ID

$login\_message = ‘You have been logged out.’;

include(‘view/login.php’);

break;

}

?>

* + Utility file that forces a valid admin user (util/valid\_admin.php) // filename for example
    - <?php

// make sure the user is logged in as administrator (or user)

if (!isset($\_SESSION[‘is\_valid\_admin’])) { // again with variable name

header(“Location: .” ); // if not logged in, redirects to the controller

}

?>

* + Code that’s included at the top of the login page
    - <?php require\_once(‘util/secure\_conn.php’); ?> // require a secure connection
  + Code that’s included at the top of other protected pages
    - <?php require\_once(‘util/secure\_conn.php’);

require\_once(‘util/valid\_admin.php’); ?> // example filename

* Setup for implementing Basic Authentication
  + Typically no logout button on view pages
  + See Murach p. 713

**Encrypting Data (other than passwords)**

* Data that needs to be encrypted and decrypted can use one of four php cryptography libraries
  + mcrypt (outdated and difficult to use correctly, being phased out)
  + Libsodium (Industry standard, and included PHP 7.2 and beyond)
  + Defuse (used by many during transition from mcrypt to Libsodium, and better than mcrypt)
  + OpenSSL
* Defuse (see p. 716 of Murach)

**Sending Emails**

* PHP function mail(); (difficult to configure and limitations)
* PHPMailer from GitHub is better
* Test Gmail Account for Development
  + Setting up a testing GMAIL account
    - Create a test account
    - Settings -> Forwarding and POP/IMAP -> POP Download -> “Enable POP for all mail” -> Save
  + Config info for GMAIL
    - Host name: smtp.gmail.com
    - Encryption: TLS
    - Port number: 587
    - Authentication: TRUE
* PHPMailer
  + Most recent version (<https://github.com/PHPMailer/PHPMailer>)
  + Must load PHPMailer class
    - Murach’s info is outdated
    - Add the PHPMailer-master folder to your project folder & require the files below (works to require within the function)
      * require('PHPMailer-master/src/PHPMailer.php');
      * require('PHPMailer-master/src/Exception.php');
      * require('PHPMailer-master/src/SMTP.php');
  + message.php file
    - <?php

function send\_email($to\_address, $to\_name, $from\_address, $from\_name, $subject, $body, $is\_body\_html = false) {

if (!valid\_email($to\_address)) {

throw new Exception(‘Recipient email address is invalid: ‘ . htmlspecialchars($to\_address));

}

if (!valid\_email($from\_address)) {

throw new Exception(‘Sender email address is invalid: ‘ . htmlspecialchars($from\_address));

if(!$mail->send()) {

throw new Exception(‘Error sending email: ‘ . htmlspecialchars($mail->ErrorInfo) );

}

require('PHPMailer-master/src/PHPMailer.php');

require('PHPMailer-master/src/Exception.php');

require('PHPMailer-master/src/SMTP.php');

// set gmail account and protocol

$mail = new PHPMailer\PHPMailer\PHPMailer();

$mail->isSMTP();

$mail->SMTPDebug = 0; // set to 2 to test, but back to 0 after

$mail->SMTPAuth = true; // new line

$mail->Host = 'smtp.gmail.com';

$mail->Port = 465;

$mail->SMTPSecure = 'ssl';

$mail->Username = "apptestemail0@gmail.com"; // app tester email acct

$mail->Password = "Gr8Pa55word!";

// set to, from, subject, and body

$mail->setFrom($from\_address, $from\_name);

$mail->addAddress($to\_address, $to\_name);

$mail->Subject = $subject;

$mail->Body = $body;

$mail->AltBody = strip\_tags($body);

if($is\_body\_html) {

$mail->isHTML(true);

}

if (!$mail->send()) {

throw new Exception('Error sending email: ' .

htmlspecialchars($mail->ErrorInfo) );

}

}

function valid\_email($email) {

if (filter\_var($email, FILTER\_VALIDATE\_EMAIL) === false) {

return false;

} else {

return true;

}

}

?>

* + Using the send\_email(); function within an ‘index.php’ case
    - This is used to allow the user to send email, can rejig to send an email to the user
    - case 'request\_query':
    - $from\_name = trim(filter\_input(INPUT\_POST, '*from\_name*')); // or combine first/last names
    - $from\_address = trim(filter\_input(INPUT\_POST, 'email', FILTER\_VALIDATE\_EMAIL));
    - $body = filter\_input(INPUT\_POST, 'body', FILTER\_SANITIZE\_SPECIAL\_CHARS); // cleanup the user input
    - $subject = 'Query Request';
    - $to\_name = 'App Developer';
    - $to\_address = 'apptestemail0@gmail.com';
    - $is\_body\_html = true;
    - try {
    - send\_email($to\_address, $to\_name, $from\_address, $from\_name, $subject,
    - $body, $is\_body\_html);
    - include 'view/success.php'; // consider making this a generic view
    - } catch (Exception $e) {
    - $error\_message = $e->getMessage();
    - include 'errors/error.php';
    - }
    - break; // p. 737-738 for implementation

**Working with Files**

* Get directory listings
  + is\_file($path); // returns true if $path exists and is a file
  + is\_dir($path); // returns true if $path eists and is a directory
  + file\_exists($path); // returns true if $path exists and is either a file or a directory
  + getcwd(); // returns a string that specifies the current working directory
    - Can use to append file names, or move up/down directories
  + Windows uses \ to separate directory levels, while Mac/Linux use /
    - Use DIRECTORY\_SEPARATOR to always insert the correct one regardless of OS
  + scandir($path); // returns an array containing a list of the files and directories in $path (returns false if not a valid dir)
  + Code to display all items in a directory or all files in a directory is listed on p. 745 Murach.
* Reading entire files
  + file($name); // returns an array with each element containing one line from the file
  + file\_get\_contents($name); // returns the contents of the file as a string
  + readfile($name); // reads a file and echoes it to the web page
* Reading/writing part of a file
  + Opening/closing files
    - fopen($path, $mode);
      * modes
        + ‘rb’ // opens the file for reading (returns false if doesn’t exist)
        + ‘wb’ // opens the file for writing, erases all data first if exists, creates the file if it doesn’t
        + ‘ab’ // opens the file for writing, new data is appended to any existing data, file is created if doesn’t exist
        + ‘xb’ // creates a new file for writing, returns false if the file already exists
    - feof($file); // returns true when the end of the specified file is reached
    - fclose($file); // closes the specified file
  + Reading/writing opened files
    - fread($file, $length); // reads up to the specified number of bytes from the specified file
    - fgets($file); // reads a line from the specified file
    - fwrite($file, $data); // writes the specified string data to the specified file (must add any new line characters you want
* Writing entire files
  + file\_put\_contents($name, $data); // writes the specified data string to the specified filename
    - Can use implode(); to convert array to a string then use file\_put\_contents to write an array of lines
* **Reading and Writing CSV Files**
  + If a field within a record contains a comma, double quote, or line break, the field must be surrounded by double quotes
  + Reading
    - fgetcsv($file); // reads one line of csv and returns them in an array
  + Writing
    - fputcsv($file, $array); // writies the specified array to the specified file as a line of csv
* **Writing a CSV file from a returned query**
  + $file = fopen(*$path*, ‘wb’); // opens a file for writing, will erase any data if there, or will create if doesn’t exist
    - // use ‘ab’ to append data to a file, or use ‘xb’ to create a new file that can’t overwrite an old file (must use a new file name)
    - $path could be ‘download.csv’
  + Need a nested array (array that contains all the data to be written, each line is an array)
  + foreach ($data as $datum) {

fputcsv($file, $datum);

}

* + fclose($file);

**Examples from flounderdb**

* + Executing queries and formatting data
    - function scdnr\_sampling\_data() {
    - global $db;
    - $query = 'SELECT \* FROM samplingdata
    - WHERE Sampler = "SCDNR"
    - ORDER BY SampleID';
    - try {
    - $statement = $db->prepare($query);
    - $statement->execute();
    - // fetchAll, using tbl primary key as array key
    - $data\_i = $statement->fetchAll(PDO::FETCH\_GROUP|PDO::FETCH\_ASSOC);
    - $statement->closeCursor();
    - // format the sampling data
    - $data = format\_sampling\_data($data\_i);
    - return $data;
    - } catch (PDOException $e) {
    - display\_error($e->getMessage());
    - }
    - }
    - function format\_sampling\_data($data\_i) {
    - $SampleIDs = get\_SampleIDs($data\_i);
    - $data = [];
    - // extract sampling data into an array
    - foreach ($SampleIDs as $SampleID) {
    - $trips = $data\_i[$SampleID];
    - $num\_trips = count($trips);
    - for ($i = 0; $i < $num\_trips; $i++) {
    - array\_push($data, $trips[$i]);
    - }
    - }
    - // set SampleID as first column
    - for ($i = 0; $i < count($data); $i++) {
    - $data[$i] = array('SampleID' => $SampleIDs[$i]) + $data[$i];
    - }
    - return $data;
    - }
    - function format\_flounder\_data($data\_i) {
    - $SampleIDs = get\_sampleIDs($data\_i);
    - //$num\_sampleids = count($SampleIDs);
    - $flounder = [];
    - $data = [];
    - // stores all data with numbered index as rows
    - foreach ($SampleIDs as $SampleID) {
    - $flounder = $data\_i[$SampleID];
    - $num\_flounder = count($flounder);
    - for ($i = 0; $i < $num\_flounder; $i++) {
    - array\_push($data, $flounder[$i]);
    - }
    - }
    - return $data;
    - }
    - function get\_all\_flounder\_data() {
    - global $db;
    - $query = 'SELECT \* FROM samplingdata s
    - INNER JOIN flounders f
    - ON s.SampleID = f.SampleID
    - INNER JOIN species sp
    - ON f.SpCode = sp.SpCode
    - ORDER BY f.SampleID';
    - try {
    - $statement = $db->prepare($query);
    - $statement->execute();
    - // fetchAll, using tbl primary key as array key
    - $data\_i = $statement->fetchAll(PDO::FETCH\_GROUP|PDO::FETCH\_ASSOC);
    - $statement->closeCursor();
    - // format flounder data
    - $data = format\_flounder\_data($data\_i);
    - return $data;
    - } catch (PDOException $e) {
    - display\_error($e->getMessage());
    - }
    - }
    - function format\_csv\_data($data, $num\_rows, $SampleIDs, $column\_headers) {
    - $array = [];
    - $array[0] = $column\_headers;
    - for ($i = 0; $i < $num\_rows; $i++) {
    - $rows = $data[$SampleIDs[$i]];
    - $array[] = $rows;
    - }
    - return $array;
    - }
    - function write\_csv($data) {
    - $file = fopen('download.csv', 'wb');
    - foreach ($data as $datum) {
    - fputcsv($file, $datum);
    - }
    - fclose($file);
    - }
    - from index.php
    - case: ‘download\_data’:
    - $file\_name = 'download.csv';
    - // output headers so the file is downloaded
    - header("Content-Type: text/csv");
    - header("Content-Disposition: attachment; filename=$file\_name");
    - // disable caching
    - header("Cache-Control: no-cache, no-store, must-revalidate"); // HTTP 1.1
    - header("Pragma: no-cache"); // HTTP 1.0
    - header("Expires: 0"); // Proxies
    - // download file
    - readfile($file\_name);
    - break;
    - Not used code, but will immediately create output.csv file from a query
    - function outputCSV($data, $file\_name = 'download.csv') {
    - // output headers so the file is downloaded
    - header("Content-Type: text/csv");
    - header("Content-Disposition: attachment; filename=$file\_name");
    - // disable caching
    - header("Cache-Control: no-cache, no-store, must-revalidate"); // HTTP 1.1
    - header("Pragma: no-cache"); // HTTP 1.0
    - header("Expires: 0"); // Proxies
    - // start the output
    - $output = fopen("php://output", "w");
    - // loop through data array
    - foreach ($data as $datum) {
    - fputcsv($output, $datum);
    - }
    - //close file
    - fclose($output);
    - }