1. PHP Basics
   * Store data outside the client’s site
   * Does not run in the browser, runs in command line (server based)
   * Created in 1994
   * Interactive shell – good for debugging, temp code (similar to console log)
   * var\_dump(Svariable name); - provides more detail about the variable, similar to console.log & debugging
   * var\_dump((bool) “variable”); - converts the variable string to a boolean value & displays the change (var\_dump)(called casting operation)
   * Empty strings & 0 – return a false Boolean
   * Interpolation requires double quotes “ , otherwise will interpret as written
   * Curly braces used inside strings to designate a value established to a variable i.e php > echo "{$firstName}\_{$lastName}";
   * Literal string – not assigned to a specific variable
   * heredoc – allows multiline strings, followed by >>> i.e.

$limerick = <<<POEM

There was a young man of Japan

Whose limericks never would scan.

When asked why this was,

He replied "It's because

I always try to fit as many syllables into the last line as ever I possibly can.

POEM;

lesson example:

Joe Blow

123 Any Street

line: php> echo “{$firstName} {$lastName)\n{$address}”;

* 1. Constants:
* values that will not change,(like API), recommended to use variables if the value will change
* Interpolation will not work with constants, will need to use concatenation
* Good practice: identify constants by using all caps, i.e. define('SIDES\_OF\_DICE', 6);
* PHP\_EOL – starts a new line in the command line, same as “\n”, but compatible command for linux, windows & mac
  1. Arrays:
* $arrayName = [variable1, 2, 3, etc];
* print\_r($arrayName); - displays the array, limited info
* var\_dump($arrayName); - displays the array & data types (int, Boolean, string, etc)
* Associative array can be a both keys & values i.e.:

php > $name = array('first' => 'John', 'last' => 'Doe');

* Same syntax as JS to access contents of arrays: (similar to JS objects)

php > echo $person[‘firstName’];

OR:

var\_dump($person);

OR:

Echo $person[‘firstName];

* Multi-dimensional arrays – seen commonly in retrieving database values

php > $people =[[‘name’ => ‘David’], [‘name’ =>‘Zach’]];

* 1. Differences between PHP & JS:
* ‘$’ – for variable names
* arrays can do more
* more built in functions
* single & double quotes are different (string interpolation – evaluating the values inside the string)
* concatenation is different
* numbers are 2 different types
* JS is in the browser, PHP is back-end language that runs on the server
* PHP syntax is more strict, command will not run without “;”
* Can cast to different data types in PHP
* PHP requires echo in the interactive shell to view or display data
* Assignment by reference
  1. Review & questions:
* Associative array syntax
* Iterating through an associative array
* How to test/run PHP

$php filename.php (in the vagrant folder command line)

filename.php (in browser – if file is saved to the public directory

* Weird symbols in the shell
* Multi-dimensional array
* Difference between object & assoc array – objects are not data structures, assoc arrays are
  1. PHP inside HTML
* Identify at the top of the HTML doc, above the <!DOCTYPE html> line
* The code is run on the server, output is sent to the browser – code will not be displayed in the browser console
* <?php – php open tag
* ?> - php closing tag
  1. Operators
* The only job for . in PHP is to concatenate the string

$name .= “ Bobberson (concatenating the new variable name)

* Post Incrementing will not show the change on the first return, will do a second echo return to see the change

php > echo $a++;

10

php > echo $a;

11

* Cannot parse together <, >, or =
  + WRONG

php > echo 1 < 2 < 0.5;

Parse error: syntax error, unexpected '<' in php shell code on line 1

* + RIGHT

php > echo (1 < 2) < 0.5;

php > echo true;

1

* If statements same as JS

$username = ‘ryano’;

$password = “awesomepassword”;

if($username == ‘ryano’ && $password == “awesome”);

* 1. EXAMPLES:

1. Write the ouput

Fwrite(STDOUT, “What is your name? “);

1. Get Input from user:

$firstName = trim (fgets(STDIN); (*Removes the white space)*

if($firstName ===’admin’) {

echo “Here are the nuclear launch codes: 12345” . PHP\_EOL;

} else {

echo “No launch codes for you! . PHP\_EOL;

}

1. Output the user’s name:

fwrite(STDOUT, “Hello $firstName\n”);

* 1. Command Line ArgumentS
     + Begin PHP file defining the number of arguments

if ($argc ===3) {

echo 'please pass the min and max for the game' . PHP\_EOL;

echo PHP\_EOL;

echo "Usage: " . PHP\_EOL;

echo " php high\_low.php <min> <max>" . PHP\_EOL;

exit;

}

* + - Define the argument number in the specific variable

$min = $argv[1];

$max = $argv[2];

* 1. Control Structures II
* Similar to javascript
  1. Array Functions
     1. Sorting
     2. Searching
     3. Push/Pop

1. PHP LIBRARY EXERCISE - INSTRUCTOR NOTES:

7/25/2017 [10:41]

If you want an automated set of tests to check your functions in library.php , do the following:

1. Save a copy of libraryTests.php to your `~/vagrant-lamp/exercises/` folder.

2. Run `vagrant ssh` to connect your terminal window into your Vagrant VM server.

3. Once inside your vagrant VM, do `cd exercises`.

4. Run `ls -la` and confirm that the downloaded `libraryTest.php` and your own `library.php` files exist in this folder.

5. Type `phpunit libraryTest.php` to run the automated tests.

* 1. Automated Test Walkthru):

define('LIBRARY\_FILE', \_\_DIR\_\_ . '/library.php'); --update with the file name for future test assessments

* Used as a pretest
* Ensures the function exists (Assertion)
* Ensures the function returns the correct data type (i.e. boolean returns true or false)

EXAMPLE:

$this->assertTrue(

function\_exists('isEven'),

"Expected isEven() function to be present."

);

1. PHP III:

MVC Basics:

* + Model View Controller – common software architecture pattern (organization/structure of the code)
  + Separation of concerns between data, data manipulation, & displaying data to the user
  + Model – code written to make connection and retrieve records from database
  + View – code written to present a user interface to the user with HTML/CSS & other front-end programs
  + Controller – manipulation of data passed back and forth between the model & the view

Web frameworks using MVC:

* Laravel for PHP
* Spring for Java
* Rails for Ruby
* Django for Python

Advantages of MVC:

* Keeps application more modular for simultaneous development
* Better organization
* Easier future development
* Multiple views for a single model

Disadvantages:

* Code can be more abstract & difficult to navigate
* More complex systems thinking

7/31/2017 Afternoon Review:

* + What happens if we put a php file outside the public directory? – will not be able to see it in the browser
  + Evaluation Model for a php/html file
    - php needs to be run on the server (rather than the local environment-dragging file into browser)
    - php files with html run from the command line will display the html content as text (how it is laid out in the html file – outside the php tags)
  + MVC – a way to separate concerns
    - Controller - Logic
    - View – display information
    - Model – the data
  + Differences between $\_GET & $\_POSTS
    - Ways to retrieve data through GET:
      * Links
      * Type directly into the url
      * Input fields
    - Post can only retrieve data in one method, through forms

1. TERMS

|  |  |
| --- | --- |
| **TERM** | **DEFINITION** |
| $variable name | Declares a variable |
| Control-C | Returns command line to the php prompt |
| Echo | Display output/show values |
| Variable Interpolation | Adding variables in between a string data, PHP will parse the interpolated variables and replace the variable with its value while processing the string data. |
| r | Array Function performed in reverse |
| k | Array function in key ascending order |
| Destructive functions | Do not maintain the key value |
| fwrite (STDOUT, ‘’ ”); | Command/call to provide an output response (space after STDOUT) |
| fgets(STDIN); | Command/call to get input from the user |
| STDIN | User input |
| STOUT | Output response (similar to echo function |
| var\_dump($argc); | ARG Count – contains count of all arguments |
| var\_dump($argv); | ARG Value - contains an array of all values passed to PHP |
| exit | Ends the argument or loop |
| die | Same as exit |
| array\_search | Searches array for a given value and returns the first key if successful, return the index of the array |
| Mixed | Mixed data type in an array search |
| mixed array\_search ( mixed $needle , array $haystack [, bool $strict = false ] ) | [, – indicates the search parameter is optional  $strict – looks for data exactly as entered (===) |
| Index [0] | If an index is 0, conditionals will interpret/default to false – use strict |
| Explode/implode | Functions to convert strings to arrays & vice versa |
| Functions | Definitions start with the function keyword, followed by the name of the function, the arguments it accepts, then a block of code to be executed.  function add($a, $b)  {  return $a + $b;  }  echo add(5, 4);  (output = 9) |
| [*substr()*](http://php.net/manual/en/function.substr.php) | Return part of a string, followed by index:  $phnumber .= substr($nameArray[1],3,3); |
| TDD | test driven development |
| array\_push() | Push one or more elements onto the end of an array |
| array\_pop() | Pop the element off the end of an array |
| array\_shift() | Shift an element off the beginning of an array  $keepsTheDocsAway = array\_shift($foods);  echo $keepsTheDocsAway;  print\_r($foods); |
| array\_unshift() | Prepend one or more elements to the beginning of an array |
| <?= | Short tag, same as <?php echo (template tag) |
| endforeach; | Closes the for each in html instead of using the closing curly bract |
| sleep(5); | Delays the code from running – the increment inside the () in seconds |
| pageController | User defined function to handle all processing for the page (part of a template) |
| @return array | An associative array of data to be used in rendering the html view |
| $\_GET | Superglobal variable to send requests and receive results based on the specific parameters designated, the get request becomes the interface – adds information to the URL |
| Query string | Single string that contains key value pairs:  Results – key, search = value  i.e. [www.youtube.com/results?search\_query=jquery](http://www.youtube.com/results?search_query=jquery) |
| Extract | Retrieves information from an array, checks each key to see whether it has a valid variable name. It also checks for collisions with existing variables in the symbol table. |
| header | Send a raw HTTP header (redirect) |

1. DATA TYPES

|  |  |  |
| --- | --- | --- |
| **DATA TYPE NAME** | **DEFINITION** | **EXAMPLE** |
| BOOLEAN | Evaluates expression to true or false | true, false |
| Float | decimal numbers | 2.0, 3.14, -5.3 |
| Integers | whole numbers | 1, 2, -5, 0 |
| Strings | characters in single or double quotes | “My name is Jennifer”, “Jennifer” |
| Arrays | List of items, indexed by number starting at 0 | [“cat”, “dog”, “turtle”, “mouse”];  cat[0], dog[1], etc. |
| Associative Arrays | Arrays with named keys | [‘first\_name => ‘Ryan, ‘last\_name’ => ‘Orsinger’]; |
| Multidimensional arrays | Arrays containing one or more arrays | $cars = array  (  array("Volvo",22,18),  array("BMW",15,13),  array("Saab",5,2),  array("Land Rover",17,15)  ); |

1. OPERATORS

Turn into note card flash cards with the operator, what the vocabulary, returns & examples of each

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **COMPARISON OPERATORS** | | | | | |
| Assuming: $x = 5, $y = null, | | | | | |
| **OPERATOR** | **NAME** | | **EXAMPLE** | | **RESULT** |
| > | Strictly greater than | | $x > 10 | | False |
| >= | Greater than or equal to | | $x >= 5 | | True |
| < | Strictly less than | | $x < -50 | | False |
| <= | Less than or equal to | | $x <= 100 | | True |
| == | Equal to (value only) | | $x == “5”  $y == undefined  True == “1”  0 == false  null == undefined  NaN == NaN | | True  True  True  True  True  False |
| != | Not equal to | | $x != “b” | | True |
| <> | Not equal type, not strict | |  | |  |
| === | Identical, equal value & type | | $x === “5”  $y === undefined | | False  False |
| !== | Not equal to value OR equal type | | $x !== “5” | | True |
| <=> | Spaceship, integer is less than or equal to, or greater than zero | |  | |  |
|  | | | | | |
| **LOGICAL OPERATORS** | | | | | |
| Assuming x=5, y=9 | | | | | |
| **OPERATOR** | **NAME** | | | **EXAMPLE** | **RESULT** |
| && | AND (both sides have to be true to be true or else result is false) | | | X<10 && x!==5 | False |
| || | OR | | | y>9 || x===5 | True |
| ! | NOT | | | !(x===y) | True |
| Values that aren’t actually true or false, are still inherently “truthy” or “falsey” values when evaluated in Boolean context | | | | | |
|  | | | | | |
| **EXPRESSIONS** | | | | | |
| **EXPRESSION EXAMPLE** | | **FUNCTION NOTES** | | | |
| $b = $a = 5 | | Assign the value five into the variable $a and $b | | | |
| $c = $a++ | | Post-increment, assign original value of $a (5) to $c | | | |
| $e = $d = ++$b | | Pre-increment, assign the incremented value of $b (6) to $d and $e (5 + 1) | | | |
| $f = double($d++); | | Assign twice the value of $d before the increment, 2\*6 = 12 to $f | | | |
| $g = double(++$e); | | Assign twice the value of $e after the increment, 2\*7 = 14 to $g | | | |
| $h = $g += 10; | | First, $g is incremented by 10 & ends with the value of 24, the value of the assignment (24) is then assigned into $h & $h ends with the value of 24 | | | |

1. CONTROL STRUCTURES

|  |  |  |
| --- | --- | --- |
| **CONDITIONALS** | | |
| Expression is evaluated to its Boolean value | | |
| **CONDITIONAL** | **PURPOSE** | **EXAMPLE** |
| if | If expression evaluates to true, PHP wil execute statement, if evaluates to false – it will ignore it | If ($a > $b)  Echo “a is bigger than b”; |
| elseif | Extends if statement to execute a different statement in case the original if evaluates to false & only if elseif conditional expression evaluates to true | if ($a > $b) {  echo "a is bigger than b";  } elseif ($a == $b) {  echo "a is equal to b"; |
| else | Extend if statement in case the expression in the if statement evaluates to false | if ($a > $b) {  echo "a is greater than b";  } else {  echo "a is NOT greater than b";  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **LOOPS** | | | |
| Repeat code in a simple statement without writing repeated lines of code (iterate) – efficient & clean iteration of a code with declarative statement to meet a conditional (i.e. $var = 1, loop may set to add one on each iteration and continue counting up until the condition is met/true , end when count <100) | | | |
| **LOOP** | **PURPOSE** | **EXAMPLE** | **RESULT** |
| while | Tells PHP to execute the nested statement(s) repeatedly, as long as the while expression evaluates to TRUE. The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time PHP runs the statements in the loop is one iteration). Sometimes, if the while expression evaluates to FALSE from the very beginning, the nested statement(s) won't even be run once | $i = 1;  while ($i <= 10) {  echo $i++; | the printed value would be  $i before the increment,(post-increment), example = 2; |
| do-while | Similar to while loops, except the truth expression is checked at the end of each iteration instead of in the beginning. The main difference from regular while loops is that the first iteration of a do-while loop is guaranteed to run (the truth expression is only checked at the end of the iteration), whereas it may not necessarily run with a regular while loop (the truth expression is checked at the beginning of each iteration, if it evaluates to FALSE right from the beginning, the loop execution would end immediately) | $i = 0;  do {  echo $i;  } while ($i > 0); | loop would run one time exactly, since after the first iteration, when truth expression is checked, it evaluates to FALSE ($i is not bigger than 0) and the loop execution ends |
| for | The first expression (expr1) is evaluated (executed) once unconditionally at the beginning of the loop. In the beginning of each iteration, expr2 is evaluated. If it evaluates to TRUE, the loop continues and the nested statement(s) are executed. If it evaluates to FALSE, the execution of the loop ends. At the end of each iteration, expr3 is evaluated (executed). Each of the expressions can be empty or contain multiple expressions separated by commas. | for ($i = 1; $i <= 10; $i++) {  echo $i;  } |  |
| foreach | Provides an easy way to iterate over arrays. foreach works only on arrays and objects, and will issue an error if used on a variable with a different data type or an uninitialized variable. There are two syntaxes:  foreach (array\_expression as $value)  statement  foreach (array\_expression as $key => $value)  statement | $arr = array(1, 2, 3, 4);  foreach ($arr as &$value) {  $value = $value \* 2;  } | $arr is now array(2, 4, 6, 8) |
| break | Ends execution of the current for, foreach, while, do-while or switch structure. Accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of. The default value is 1, only the immediate enclosing structure is broken out of | $arr = array('one', 'two', 'three', 'four', 'stop', 'five');  while (list(, $val) = each($arr)) {  if ($val == 'stop') {  break;  }  echo "$val<br />\n";  } |  |
| continue | Used within looping structures to skip the rest of the current loop iteration & continue execution at the condition evaluation & then the beginning of the next iteration. Accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of. The default value is 1, thus skipping to the end of the current loop. | while (list($key, $value) = each($arr)) {  if (!($key % 2)) { // skip even members  continue;  }  do\_something\_odd($value);  }  $i = 0;  while ($i++ < 5) {  echo "Outer<br />\n";  while (1) {  echo "Middle<br />\n";  while (1) {  echo "Inner<br />\n";  continue 3;  }  echo "This never gets output.<br />\n";  }  echo "Neither does this.<br />\n";  } |  |
| switch | Similar to a series of IF statements on the same expression, to compare the same variable (or expression) with many different values, & execute a different piece of code depending on which value it equals to. The switch statement executes line by line (actually, statement by statement). In the beginning, no code is executed. Only when a case statement is found whose expression evaluates to a value that matches the value of the switch expression does PHP begin to execute the statements. PHP continues to execute the statements until the end of the switch block, or the first time it sees a break statement. If you don't write a break statement at the end of a case's statement list, PHP will go on executing the statements of the following case. | switch ($i) {  case 0:  echo "i equals 0";  break;  case 1:  echo "i equals 1";  break;  case 2:  echo "i equals 2";  break;  } |  |

1. CURRICULUM EXERCISES

|  |  |
| --- | --- |
| **CURRICULUM EXERCISE** | **SOLUTION** |
| Create another for loop that counts from 1 to 100, but stops after 10 using break | or ($i = 1; $i <= 100; $i++) {  if ($i == 11) {  break;  }  echo $i . PHP\_EOL;  } |
| Create a do-while loop that will count by 2's starting with 0 and ending at 100. Follow each number with a newline | $a = 0;  do {  echo $a .PHP\_EOL;  $a += 2;  } while ($a <= 100); |
| Alter your loop to count backwards by 5's from 100 to -10 | $a = 100;  do {  echo $a .PHP\_EOL;  $a -= 5;  } while ($a >= -10); |
| Create a do-while loop that starts at 2, & displays the result $a \* $a on each line while $a is less than 1,000,000. Output should equal:  2  4  16  256  65536 | $a = 2;  do {  echo $a .PHP\_EOL;  $a \*= $a;    } while ($a < 1000000); |
| Write a program that prints the numbers from 1 to 100. But for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers which are multiples of both three and five print “FizzBuzz”. | for ($number = 1; $number <= 100; $number++) {  if (0 == ($number % 3) && ($number % 5)) {  echo "fizzbuzz" . PHP\_EOL;  }  else if (0 == ($number % 3)){  echo "fizz" . PHP\_EOL;  }  else if (0 == ($number % 5)){  echo "buzz" . PHP\_EOL;  }  else {  echo "$number" . PHP\_EOL;  }  } |
| for (){} // contains: initialization, condition to check, & expression to evaluate(incement usually)  $a = 1; //initialization  while ($a <= 5) //the condition  echo "\$a has a value of {$a}\n";  $a++; //increment  }  //while loop:  for ($a = 1; $a <=5; $a++) {  echo "\$a has a value of {$a}\n";  } | |
| function makeTitleCase($str)  {  return ucfirst(strtolower($str));  }  echo makeTitleCase("aBc");  function greeting($firstName, $lastName)  {  $output = "";  $output = "$firstName $lastName";  if (is\_string($firstName) && is\_string($lastName)) {  $output = "Hello, $firstName $lastName !";  } else {  $output = "Sorry two valid names were not given";  }  return $output;  } | |
| // IF AND OR:  $x = 0;  $y = 5;  $z = 10; | If ($x < $y && $y < $z){  echo "{$x} < {$y} < {$z}\n";  }  If (0 < $x || $x < 10) {  echo "0 is less than {$x} OR {$x} is less than 10";  }  if (0 < $y || $y < 10) {  echo "0 is less than {y} OR {y} is less than 10";  }  if (0 < $z || $z < 10) {  echo "0 is less than {z} OR {z} is less than 10";  } |
| If 0 is less than $x AND $x is less than 10, then echo the result as a sentence "0 is less than {$x} AND {$x} is less than 10". | if (0 < $x && $x <10) {  echo "0 is less than {$x} AND {$x} is less than 10";  } |
| Repeat the if statement for $y and $z. | if (0 < $y && $y <10) {  echo "0 is less than {$y} AND {$y} is less than 10";  }  if (0 < $z && $z <10) {  echo "0 is less than {$z} AND {$z} is less than 10";  } |
|  |  |

1. SAMPLE PROBLEM SOLVING IN PHP

|  |  |  |
| --- | --- | --- |
| **PROBLEM** | **EXPLANATION** | **SOLUTION** |
| Write a function called isEven that takes in a variable and returns true or false if the provided input is evenly divisible by two or not | * Use $input to enable the function to be used multiple times as the variable is not hard coded * Is\_numeric – only run code if input is a numeric value * If input is divisible by 2, then number is even * Otherwise, it is odd and returns a false | function isEven($input)  {  if(is\_numeric($input)) {  return $input % 2 === 0;  }  return false;  } |
| Write a function called isVowel that returns true or false if the provided input is the letter 'a', 'e', 'i', 'o', or 'u' | * If input is a not a string value & character count equal to 1, then return false * Convert string to lower case * If input value & type is equal to “a”, “e”, “I”, “o” OR “u”, then return true * Otherwise return false | function isVowel($input)  {  if(!is\_string($input) && count($input) == 1)  return false;  }  $input = strtolower($input);  if($input === 'a' || $input === 'e' || $input === 'i' || $input === 'o' || $input === 'u') {  return true;  } else {  return false;  }  } |
| Write a function called "first" that takes in an argument that could be an array or a string. Return the first character if the argument is a string. Return the first element of the array if the input is an array. | If input is string or array, return the 1st character or index item[0], of the input | function first($input)  {  if(is\_string($input) || is\_array($input)) {  return $input[0];  }  } |
| Write a function called "second" that takes in input that could be an array or a string. The function should return the second character or element of the input. | If input is string or array, returns the 2nd character or index item[1] of the input | function second($input)  {  if(is\_string($input) || is\_array($input)) {  return $input[1];  }  } |
| Write a function called "last" that takes in an input that could be an array or a string. Function should return the last character or element of the input. | * If input is string, return the last character of the string(-1) * If input is array, count array items and return the last array item (-1); | function last($input){  if(is\_string($input) {  return $input[(strln($input)-1)];  }  if (is\_array($input)) {  return $input[(count($input)-1)];  } |
| Write a function called "reverse" that takes in an input of either a string or an array. Function should return the elements or characters in reverse order. | * If input is string, return string content in reverse order * If input is array, return output of array items in reverse order | function reverse($input)  {  if(is\_string($input)) {  return strrev($input);  }  if(is\_array($input)) {  return array\_reverse($input);  }  } |
| Write a function called "random" that takes in an input that can be either a string or an array. Function should return a random element or character from the input. | * If input is string, random index will measure the length of the string & return 1 random character of the string * If input is array, return random array item, defined at the $key element | function random($input)  {  if(is\_string($input)){  $randomIndex = mt\_rand(0, strlen($input)-1);  return $input[$randomIndex];  }  if(is\_array($input)){  $key = $array\_rand($input);  return $input[$key];  }  } |
| Write a function named isNegative that accepts a number and returns true or false based on whether the input is negative | * Is\_numeric validates if input is not a number & returns false if not a number = true   Otherwise, validates if input number is less than 0, if less than 0 – returns true, if greater than 0 returns false – based on the arguments passed below | function isNegative($input)  {  if(!is\_numeric($input)) {  return false;  }  return $input < 0;  }  PASSED ARGUMENTS:  isNegative(-1); // true  isNegative(6); // false |
| Write a function named average that accepts an array of numbers and returns the average of those numbers | * !Is\_array –determines input is not an array, returns false * Foreach to pull value from the array input key * !Is\_numeric – determines input value is not a number & returns false;   otherwise calculates & returns the average by taking the array sum of the array input keys & dividing it by the total count of the array items | function average($input)  {  if(!is\_array($input)) {  return false;  }  foreach($input as $value) {  if(!is\_numeric($value)) {  return false;  }  }  return array\_sum($input) / count($input);  }  PASSED ARGUMENTS:  average([1, 2, 3]) // 2  average([4, 6, 8, 10, 12]) // 8  ///with loop:  $sum = 0;  foreach($input as $value) {  $sum += $value;  }  return $sum / count($input); |
| Write a function named countOdds that accepts an array of numbers and returns the number of odd numbers in the array | * set variable count to 0 * is array determines if input is an array & returns false * for each looks at the value for the input * is\_numeric gets value only if input is a number and the value of the number is not divisible by 0 to determine input is a number and an odd number * $count++ increments the value only after the foreach evaluation is true   return the count of the passed input arguments meeting the conditional | function countOdds($input)  {  $count = 0;  if(is\_array($input)){  return false;  }  foreach($input as $value) {  if(is\_numeric($value) && $value % 2 !== 0) {  $count++;  }  }  return $count;  }  PASSED ARGUMENTS:  countOdds([1, 2, 3]) // 2  countOdds([4, 6, 8, 10]) // 0  countOdds([1, 2, 1, 1]) // 3 |
| Write a function named convertNameToAssociativeArray that accepts a string that contains a first name and last name separated by a space, and returns an associative array with keys of firstName and lastName. | * !is\_string – determines input is not a string & returns false * otherwise takes string & explodes data into an empty array * further defines the array by defining the key elements within the array and index   then returns the input as an array | function convertNameToAssociativeArray($input)  {  if(!is\_string($input)) {  return false;  }  $inputArray = explode(",", $input);  $outputArray = [];  $outputArray['firstName'] = $inputArray[0];  $outputArray['lastName'] = $inputArray[1];  return $outputArray;  }  PASSED ARGUMENTS:  convertNameToAssociativeArray('Harry Potter') // ['firstName' => 'Harry', 'lastName' => 'Potter']  convertNameToAssociativeArray('Ron Weasley') // ['firstName' => 'Ron', 'lastName' => 'Weasley'] |
| Write a function named fiveTo that accepts a number and returns an array of the integers from 5 to (including) the passed number. | * !is\_numeric – determines input is not a number & returns false * sets output array to an empty array * if $1>=5 – sets conditional if input is greater than or equal to 5, for each input greater than 5,increment up output numbers beginning with 5 up to the input number (input is 10, return 5, 6, 7, 8, 9, 10) * for each input that is less than 5, output 5 and decrement down to list each number from 5 to the input (if input is -2, then it will return 5, 4, 3, 2, 1, 0, -1, -2) * Return output as defined in the conditionals   Simpler solution:   * !is\_numeric to determine if input is not numeric and return false   otherwise, uses built-in PHP function to return range, with parameter of 5 & input | function fiveTo($input)  {  if(!is\_numeric($input)) {  return false;  }  $outputArray = [];  if($i >= 5) {  for($i = 5; $i <= $input; $i++){  $outputArray[] = $i;  }  }  if ($input < 5) {  for($i = 5; $i >= $input; $i --) {  $outputArray[] = $i;  }  }  return $outputArray;  }  //SIMPLER SOLUTION:  function fiveTo($input)  {  if(!is\_numeric($input)) {  return false;  }  return range(5, $input);  }  PASSED ARGUMENTS:  fiveTo(10) // [5, 6, 7, 8, 9, 10]  fiveTo(6) // [5, 6]  fiveTo(5) // [5]  fiveTo(-2) // [5, 4, 3, 2, 1, 0, -1, -2] |
| Write a function named upperCaseLastNames that accepts an array of associative arrays where each associative array has indices firstName and lastName and returns the same array of associative arrays with each last name capitalized | * !is\_array – to determine input is not array & return false * for each to pull in value of the array key, index “lastName * ucfirst – to retrieve the value of the key index lastName and convert the first character to uppercase   Return the input as outlined in the foreach | function upperCaseLastNames($input)  {  if(!is\_array($input)) {  return false;  }  foreach ($input as $key => $person) {  $person['lastName'] = ucfirst($person['lastName']);  $input[$key] = $person;  }  return $input;  }  PASSED ARGUMENTS:  $names = [  ['firstName' => 'Harry', 'lastName' => 'Potter'],  ['firstName' => 'Ron', 'lastName' => 'weasley'],  ['firstName' => 'Hermione', 'lastName' => 'granger'],  ];  upperCaseLastNames(names);    [  ['firstName' => 'Harry', 'lastName' => 'Potter'],  ['firstName' => 'Ron', 'lastName' => 'Weasley'],  ['firstName' => 'Hermione', 'lastName' => 'Granger'],  ] |
| Create a for loop that shows all even numbers between 1 and 100 using continue. | * $i = 1 – sets the starting variable equal to 1 as specified in instructions * $1 <= 100, sets the loop to each once it reaches 100 * $i++ --post increment (iterate up) from the variable of 1 * if($i % 2 != 0) – if incremented value is not divisible by 2 continue the loop to the next value & only display if the incremented value is divisible by 2, | for ($i = 1; $i <= 100; $i++) {  if ($i % 2 != 0) {  continue;  }  echo "$i" . PHP\_EOL;  } |

1. Reading/writing Files

|  |  |  |
| --- | --- | --- |
| **COMMAND** | **DEFINITION /NOTES** | **EXAMPLE** |
| $filename | Identifies the specific resource  Ensure in the same folder path or that menu path is identified | 'cities.txt'; |
| ‘w’ | Mode – open for writing only, place file pointer at the beginning of file & truncate file to zero length  Writes over content in file – should be used with caution |  |
| ‘a’ | Mode – open for writing only, place file pointer at the end of the file, if file does not exist – attempts to create file –add to existing content |  |
| $handle | abstract reference to a resource | $handle = fopen($filename, 'r'); |
| 'r' | Mode - read the file/resource |
| filesize($filename) | * Returns the number of bytes(characters long) as filesize; * Best practice to include the number of characters, without it specified it will read the entire file, reads the designated number starting at beginning of file * Must be included in the specified order of handle, filesize & filename | $contents = fread($handle, filesize($filename)); |
| $contents = trim($contents); | Eliminate whitespace, use for best practice |  |
| echo $contents; | Display file contents |  |
| $contentsArray = explode("\n", $contents); | Instructs the content to be displayed with a delimiter of new line ("\n") |  |
| print\_r($contentsArray); |  |  |
| fclose($handle); | Required to instruct code to close after data collected (releases memory used to keep it open?) |  |
|  |  |  |
|  |  |  |

PHP I QUIZ

PHP runs on the virtual machine or server while javascript runs in the browser.

True

Explanation: PHP is a server side language

False

What is the command to open the PHP interactive shell?

php -A

php --shell

php -a

Explanation: lowercase a is the correct flag, if you forget use php -h to get the help menu

$message = 'Good luck on the quiz!';

Given the variable above, what can we use to display $message's content?

echo

print\_r

var\_dump

printf

All of the above

Explanation: All three can be used to output the content, print\_r and var\_dump give us a little more information as well

What error will the following code produce?

$array = ['this','quiz is','super sonic'];

echo $array;

Notice: Array to string conversion

Explanation: We cannot echo an array like this. we would need to loop through the array and echo its contents individually

Parse error: syntax error, unexpected ';', expecting ')'

no error, the code will execute fine.

Explanation: it will error with an array to string conversion error

What is the concatenation operator is PHP?

.

Explanation: not to be confused with + in JavaScript

+

All of the above

What will the following code output?

var\_dump( ((true || false) && (false || !false )) );

true

bool(true)

Explanation: var\_dump also gives us the type of what value it is given

false

bool(false)

What will the following code output?

$num = 3;

var\_dump($num++);

int(3)

Explanation: since we used a post-incrementor the var\_dump gives us 3 even though $num at the end is 4

int(4)

3

4

What does '42' <> 42 equate to?

false

Explanation: <> is the same as !=, and the string '42' does loosely equal the int 42

true

Explanation: <> is the same as !=, and the string '42' does loosely equal the int 42

PHP II

Which of the following is not valid syntax for an array in PHP?

$b = ['orange', 'mango'];

$c = array('apple', 'banana');

All of the above

All of the below

Explanation: example $a, has no keyword array and $d has an extra keyword new

$a = ('strawberry', 'pineapple');

$d = new array ('kiwi', 'blueberry');

What would echo $argv[2]; display on if we ran the following command? (assume the echo is in the file test.php)

php test.php 23

23

NULL

PHP Notice: Undefined offset: 2

Explanation: since we only gave 1 extra argument, $argv[2] is not defined only index 0 and 1 are defined

What will the following code output?

for ($i = 1; $i <= 15 ; $i+) {

if ($i / 15 == 0) {

echo "FizzBuzz" . PHP\_EOL;

} else if ($i / 3 == 0) {

echo "Fizz" . PHP\_EOL;

} else if ($i / 5 == 0) {

echo "Buzz" . PHP\_EOL;

} else {

echo $i . PHP\_EOL;

}

}

1(forever)

1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz

PHP Parse error: syntax error, unexpected ')'

Explanation: This happens because we used on + instead of 2 for our incrementor

krsort() sorts by key and maintains key association.

true

Explanation: only sort(), rsort() and shuffle() don't maintain key association, and the k in a sort function name means it sorts by key

false

array\_search() is case sensitive.

true

Explanation: to array\_search() capital letters are not the same as their lowercase counterpart

false

Which snippet will remove 'jagged little pill' from the array?

$best\_songs = [

'call me maybe',

'auld lang syne',

'jagged little pill',

'moon river'];

array\_splice($best\_songs, array\_search('jagged little pill', $best\_songs), 1);

array\_splice($best\_songs,2,1);

unset($best\_songs[2]);

All of the above

Explanation: all of these will remove the desired string

When using fopen($filename, 'w');, what does w make the function do assuming the file exists?

Opens the file and puts the pointer at the beginning of the file

Opens the file, puts the pointer at the beginning and truncates the file to zero length

Explanation: if the file had not been found, it would have created it

Opens the file and puts the pointer at the end of the file

Assuming this code will not cause an error, what will break 3; do?

breaks out of the loop we are currently in and 2 parent loops

Explanation: the number tells us how many of the loops we will break out of, limit is how many loops we are currently inside of

breaks out of the current loop and skips the next 3 iterations of the parent loop

Why do we put php tags in our php scripts (that is <?php)?

While not necessary, it is a best practice to indicate where our php code runs.

They are necessary in order for our php code to be interpreted as php.

To help maintain a clean code formatting.

The tags, while not necessary, are an offering to the php deity that reduces the risk of bugs in our code.