1. TERMS

|  |  |
| --- | --- |
| **TERM** | **DEFINITION** |
|  |  |
|  |  |
|  |  |
| 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 |

1. PHP Basics
   1. Store data outside the client’s site
   2. Will not run in the browser
   3. Created in 1994
   4. Interactive shell – good for debugging, temp code (similar to console log)
   5. Float numbers = decimal numbers
   6. Integers = whole numbers
   7. “Echo” – used to output/show values
   8. $variable name; - declares a variable
   9. var\_dump(Svariable name); - provides more detail about the variable, similar to console.log & debugging
   10. Control-C = returns to the php prompt
   11. var\_dump((bool) “variable”); - converts the variable string to a boolean value & displays the change (var\_dump)(called casting operation)
   12. Empty strings & 0 – return a false Boolean
   13. Interpolation requires double quotes “ , otherwise will interpret as written
   14. Curly braces used inside strings to designate a value established to a variable i.e php > echo "{$firstName}\_{$lastName}";
   15. Literal string – string not assigned to a specific variable
   16. 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}”;

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

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’]];

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

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

**Data Types:**

* booleans => true, false
* float => decimal numbers , i.e. 2.0, 3.14, -5.3
* integers => whole numbers: i.e. 1, 2, -5, 0
* strings => characters in single or double quotes
* arrays => [1, 2, 3, 4]
* associative\_arrays = [‘first\_name => ‘Ryan, ‘last\_name’ => ‘Orsinger’];
* define (“PI”, 3.141);

**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”

**Front End vs Back-End**

Data Control Structures

I/O Streams

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