# Everlytic Developer Assessment

|  |  |
| --- | --- |
| Name | Michael Townshend |

|  |  |
| --- | --- |
| Date | 25 September 2020 |

Please complete the answers to the questions below. The assessment should take roughly 30 minutes.

### What is the difference between public, protected and private in a class definition?

|  |
| --- |
| These three keywords determine the scope of a property or method in a class. Private properties/constants/methods care only accessible within the scope of the class. Protected properties/constants/methods are available within the class and any classes that descend from that class. Public properties/constants/methods are accessible from anywhere. |
|  |
|  |

### Given this code: function doSomething(&$foo) { $bar = $foo; $foo += 1; return $foo; } $value = 3; $result = doSomething($value); echo "value: $value, result: $result"; What will be output to screen and why?

|  |
| --- |
| value: 4, result: 4 |
| $foo is passed by reference using the & in the parameter. This means that even though $foo += 1 is inside the function it still updates $value by 1 because it is referred to, not assigned. The total, now 4 is assigned to $result. |
|  |

### What is wrong with this query: "SELECT \* FROM table WHERE id = $\_POST[ 'id' ]"?

|  |
| --- |
| <https://xkcd.com/327/> |
| Never, ever, ever, ever pass user input directly to a query until you have sanitized it and verified it. And that’s before we start talking about backpacking the table names and associating them with a schema. |
|  |

### What is the cause of this warning: 'Warning: Cannot modify header information - headers already sent', and what is a good practice to prevent it?

|  |
| --- |
| This means that there is a call to a header() function in the code. This function modifies the header information but headers are sent with the first response from the script. This can be caused by an error, deliberate output, rogue spaces, byte order markers and other invisible characters. Once the headers have been sent to the user, they can no longer be modified and so an error is thrown. |
| This error can be avoided by ensuring your code is well structured, not printing errors (by turning off error\_reporting - not great for debugging but essential for production), or by using output buffering. |
|  |

### What is wrong with this code: class Foo { protected $bar; public function \_\_construct() { $this->bar = 1; } public static function doSomething() { return $this->bar; } }

|  |
| --- |
| static functions have no access to $this so PHP throws a fatal error. You can get around this by passing the protected variable as a parameter in the static method but it is generally a very bad idea and it will almost certainly haunt you. |
|  |
|  |

### 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($i=1; $i <= 100; $i++) {

if(0 == $i % 15) echo "<p>FizzBuzz</p>";

elseif(0 == $i % 5) echo "<p>Buzz</p>";

elseif(0 == $i % 3) echo "<p>Fizz</p>";

else echo "<p>$i</p>";

}

### What does the following code do? Explain what’s going on there.

$date = '08/26/2003';

print preg\_replace('/([0-9]+)\/([0-9]+)\/([0-9]+)/'‚ '$2/$1/$3', $date);  
preg\_replace uses PCRE regular expressions to search a string for a pattern and then replace it with something. In this instance, it looks for one or more digits which it captures in to a group, followed by a slash followed by one or more digits captured in to a second group followed by a slash and then one or more digits captured in to a third group. This replace function then just reorders the groups. It turns a proper date in to an American one.

|  |
| --- |
| You could replace [0-9] with \d if you like. |
|  |

### Given a line of text $string, how would you write a regular expression to strip all the HTML tags from it?

|  |
| --- |
| The most accurate answer is that you wouldn’t. HTML can be so complex and with such huge scope in attributes it would not be hard to create a string that fails any regex. There are so many HTML parser and PHP classes that can do a much better job than any single regex, it is much, much better to use one of those. |
| But, **/<[^>]\*>/g** is generally considered to be the most agreeable solution. |
|  |

### A palindrome is a word that reads the same backward or forward. Write a function that checks is a given word is a palindrome. Characters case should be ignored. EG. Deleveled is a palindrome and should return true as character case is ignored.

### <?php class Palindrome

### {

### public static function isPalindrome($word)

{

return (strrev(strtolower($word)) == strtolower($word)) ? true : false;

}

### } echo Palindrome::isPalindrome('Deleveled');

If this was a production method I’d probably be a bit more verbose and readable. I’d also do error checking. But, I thought this single liner was neat.

### What security issue is prevalent in the code below and how would you fix it? <?php $messageStmt = $db->query('select message\_text from messages where message\_id = 1'); $messageStmt->execute(); $message = $messageStmt->fetch(PDO::FETCH\_OBJ); ?> <div><?= $message->message\_text; ?></div> It is bad practice to pass data directly to a query, especially if this data may have come from a user. PDO prepared statements help prevent SQL injection attacks. Rewriting the first two lines as: $messageStmt = $db->query('SELECT `message\_text` FROM `messages` WHERE `message\_id` = :message\_id'); $messageStmt->execute(['message\_id'] => 1); will fix the security issue. We also mustn’t underestimate the role played by writing our SQL using best practice with case and backpacks. IT ensures our code is more readable and so less prone to bugs and errors but also less likely to fail if we inject a reserved word. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Write an inner join for the following tables Picture 6

|  |
| --- |
| SELECT \* FROM `User`  INNER JOIN `Address`  ON `User`.`UserKey` = `Address`.`Usrkey`  WHERE .... |
| Inner joins are the default for MySQL so you could just write … JOIN `Address` but I think the full join type helps to make the code more readable, particularly with very complex queries. Also, depending on the greater structure of the data and these tables, there may be instances where we have Users who don’t have an address. In this case, and LEFT JOIN may be more appropriate. |

### 