1

The output will be as follows:

```
5
11
7
1
5
```

Here's are the two key facts that explain why:

1. The term $x++ says to use the current value of $x and *then* increment it. Similarly, the term $x-- says to use the current value of $x and *then* decrement it.
2. The increment operator (++) has higher precedence then the sum operator (+) in order of operations.

With these points in mind, we can understand that $x+++$x++ is evaluated as follows: The first reference to $x is when its value is still 5 (i.e., *before* it is incremented) and the second reference to $x is then when its value is 6 (i.e., before it is *again* incremented), so the operation is 5 + 6 which yields 11. After this operation, the value of $x is 7 since it has been incremented twice.

Similarly, we can understand that $x---$x-- is evaluated as follows: The first reference to $x is when its value is still 7 (i.e., *before* it is decremented) and the second reference to $x is then when its value is 6 (i.e., before it is *again* decremented), so the operation is 7 - 6 which yields 1. After this operation, the value of $x is back to its original value of 5, since it has been incremented twice and then decremented twice.

2

var_dump(0123 == 123) will output bool(false) because the leading 0 in 0123 tells the PHP interpreter to treat the value as octal (rather than decimal) value, and 123 octal is equal to 83 decimal, so the values are not equal.

var_dump('0123' == 123) will output bool(true) since the string 0123 will automatically be coerced to an integer when being compared with an integer value. Interestingly, when this conversion is performed, the leading 0 is ignored and the value is treated as a decimal (rather than octal) value, so the values are bother 123 (decimal) and are therefore equal.

var_dump('0123' === 123) outputs bool(false) since it performs a more strict comparison and does not do the automatic type coercion of the string to an integer.

3

Surprisingly to many, the above code will output bool(true) seeming to imply that the and operator is behaving instead as an or.

The issue here is that the = operator takes precedence over the and operator in order of operations, so the statement $x = true and false ends up being functionally equivalent to:

```
$x = true;       // sets $x equal to true
true and false;  // results in false, but has no affect on anything
```

This is, incidentally, a great example of why using parentheses to clearly specify your intent is generally a good practice, in any language. For example, if the above statement $x = true and false were replaced with $x = (true and false), then $x would be set to false as expected.

4

The correct answer is 18.

Here's why:

PHP supports automatic type conversion based on the context in which a variable or value is being used.

If you perform an arithmetic operation on an expression that contains a string, that string will be interpreted as the appropriate numeric type for the purposes of evaluating the expression. So, if the string begins with one or more numeric characters, the remainder of the string (if any) will be

ignored and the numeric value is interpreted as the appropriate numeric type. On the other hand, if the string begins with a non-numeric character, then it will evaluate to zero.

With that understanding, we can see that "15%" evaluates to the numeric value 15 and "$25" evaluates to the numeric value zero, which explains why the result of the statement $x = 3 + "15%" + "$25" is 18 (i.e., 3 + 15 + 0).

Please note that as of PHP 7.2, this code produces an error.

5

The result of var_dump(PHP_INT_MAX + 1) will be displayed as a double (in the case of this specific example, it will display double(9.2233720368548E+18)). *The key here is for the candidate to know that PHP handles large integers by converting them to doubles (which can store larger values).*

And interestingly, the result of var_dump((int)(PHP_INT_MAX + 1)) will be displayed as a negative number (in the case of this specific example, it will display int(-9223372036854775808)). *Again, the key here is for the candidate to know that the value will be displayed as a negative number, not to know the precise value.*