Toptal®

21 Essential PHP Interview Questions *

What are the differences between `echo` and `print` in PHP?

Hide answer

`echo` and `print` are largely the same in PHP. Both are used to output data to the screen.

The only differences are as follows:

1. `echo` does not return a value whereas `print` does return a value of 1 (this enables `print` to be used in expressions).
2. `echo` can accept multiple parameters (although such usage is rare) while print can only

What will this code output and why?

```
$x = true and false;
var_dump($x);
```

Toptal®

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

What will be the output of the code below and why?

```
$x = 5;
echo $x;
echo "<br />";
echo $x+++$x++;
echo "<br />";
echo $x;
echo "<br />";
echo $x---$x--;
echo "<br />";
echo $x;
```

Hide answer

The output will be as follows:

```
5
11
7
1
5
```

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

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

What will be the values of `$a` and `$b` after the code below is executed? Explain your answer.

```
$a = '1';
$b = &$a;
$b = "2$b";
```

Hide answer

Both `$a` and `$b` will be equal to the string `"21"` after the above code is executed.

Here's why:

The statement `$b = &$a;` sets `$b` equal to a *reference* to `$a` (as opposed to setting `$b` to the then-current *value* of `$a`). Thereafter, as long as `$b` remains a reference to `$a`, anything

Toptal®

What will be the output of each of the statements below and why?

```
var_dump(0123 == 123);
var_dump('0123' == 123);
var_dump('0123' === 123);
```

Hide answer

`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

What is the problem with the code below? What will it output? How can it be fixed?

```
$referenceTable = array();
$referenceTable['val1'] = array(1, 2);
$referenceTable['val2'] = 3;
$referenceTable['val3'] = array(4, 5);

$testArray = array();

$testArray = array_merge($testArray, $referenceTable['val1']);
var_dump($testArray);
$testArray = array_merge($testArray, $referenceTable['val2']);
var_dump($testArray);
$testArray = array_merge($testArray, $referenceTable['val3']);
var_dump($testArray);
```

```
NULL
NULL
```

You may also see two warnings generated, similar to the following:

```
Warning: array_merge(): Argument #2 is not an array
Warning: array_merge(): Argument #1 is not an array
```

The issue here is that, if *either* the first or second argument to `array_merge()` is not an array, the return value will be `NULL`. For example, although one might reasonably expect that a call such as `array_merge($someValidArray, NULL)` would simply return `$someValidArray`, it instead returns `NULL`! (And to make matters worse, this is not documented well at all in the PHP documentation.)

As a result, the call to `$testArray = array_merge($testArray, $referenceTable['val2'])` evaluates to `$testArray = array_merge($testArray, 3)` and, since `3` is not of type `array`, this call to `array_merge()` returns `NULL`, which in turn ends up setting `$testArray` equal to `NULL`. Then, when we get to the next call to `array_merge()`, `$testArray` is now `NULL` so `array_merge()` again returns `NULL`. (This also explains why the first warning complains about argument #2 and the second warning complains about argument #1.)

The fix for this is straightforward. If we simply typecast the second argument to an `array`, we will get the desired results. The corrected `array_merge()` calls would therefore be as follows:

```
$testArray = array_merge($testArray, (array)$referenceTable['val1']);
var_dump($testArray);
$testArray = array_merge($testArray, (array)$referenceTable['val2']);
var_dump($testArray);
$testArray = array_merge($testArray, (array)$referenceTable['val3']);
var_dump($testArray);
```

which will yield the following output (and no warnings):

```
array(2) { [0]=> int(1) [1]=> int(2) }
array(3) { [0]=> int(1) [1]=> int(2) [2]=> int(3) }
array(5) { [0]=> int(1) [1]=> int(2) [2]=> int(3) [3]=> int(4) [4]=> int(5) }
```

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).

After the code below is executed, what will be the value of `$text` and what will `strlen($text)` return? Explain your answer.

```
$text = 'John ';
$text[10] = 'Doe';
```

Hide answer

After the above code is executed, the value of `$text` will be the string "`John  D`" (i.e., "John", followed by six spaces, followed by "D") and `strlen($text)` will return `11`.

There are two things going on here.

First of all, since `$text` is a string, setting a single element of `$text` simply sets that single character to the value specified. The statement `$text[10] = 'Doe'` therefore sets that single character to `'D'` (i.e., the first character in the string `"Doe"`, since an element of a string can only be a single character).

Secondly, `$text[10] = 'Doe'` says to set the 11th character of the string (remember that indices are zero-based) to `'D'`. Prior to that statement, though, the length of the string `$text`

integer value (value is based on the version of PHP being run and the platform it is running on).
Assume that `var_dump(PHP_INT_MAX)` will yield `int(9223372036854775807)`.
In that case, what will be the result of `var_dump(PHP_INT_MAX + 1)`? Also, what will be the result of `var_dump((int)(PHP_INT_MAX + 1))`?
NOTE: It's not important to supply the exact value when answering the question, but rather to explain what will happen and why.

Hide answer

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*

What does the follow code echo?

```
$a = "PHP";
$a = $a + 1;
echo $a;
```

Hide answer

The number `1`.

Note, in PHP 7.2 this throws a warning that a non-numeric value is encountered on the line and does not do the conversion to the `int`, so it echoes `PHP` instead.

```
        '1' => 'Z10',
        '2' => 'z12',
        '3' => 'Z2',
        '4' => 'z3',
)
```

After sorting, should become:

```
array(
        '0' => 'z1',
        '3' => 'Z2',
        '4' => 'z3',
        '1' => 'Z10',
        '2' => 'z12',
)
```

Hide answer

The trick to solving this problem is to use three special flags with the standard `asort()` library function:

```
asort($arr, SORT_STRING|SORT_FLAG_CASE|SORT_NATURAL)
```

The function `asort()` is a variant of the standard function `sort()` that preserves the index association. The three flags used above `SORT_STRING`, `SORT_FLAG_CASE` and `SORT_NATURAL` forces the sort function to treat the items as strings, sort in a case-insensitive way and maintain natural order respectively.

Note: Using the `natcasesort()` function would not be a correct answer, since it would not

What are Traits?

Hide answer

Traits are a mechanism that provides some of the reuse advantages of multiple inheritance in

Toptal

Hide answer

PEAR (**P**HP **E**xtension and **A**pplication **R**epository) is a framework and repository for reusable PHP components. PEAR is a code repository containing all kinds of php code snippets and libraries.

PEAR also offers a command-line interface that can be used to automatically install packages.

What is use of the `header()` function in PHP?

Hide answer

1. `header()` is used to redirect from one page to another: `header("Location: index.php");`
2. `header()` is used to send an HTTP status code: `header("HTTP/1.0 this Not Found");`
3. `header()` is used to send a raw HTTP header: `header('Content-Type: application/json');`

Consider the following code:

```
$x = NULL;

if ('0xFF' == 255) {
    $x = (int)'0xFF';
}
```

What will be the value of `$x` after this code executes? Explain your answer.

Hide answer

Perhaps surprisingly, the answer is neither NULL nor 255. Rather, the answer is that `$x` will equal 0 (zero).

(since the other operand is an integer). So in this case, '0xFF' is converted to its integer equivalent which is 255. Since 255 = 255, this condition evaluates to true. (Note that this only works for hex strings, not for octal or binary strings.)

But if that's the case, shouldn't the statement `$x = (int)'0xFF';` execute and result in `$x` being set equal to 255?

Well, the statement does execute, but it results in `$x` being set equal to 0, *not* 255 (i.e., it is *not* set to the integer equivalent of '0xFF'). The reason is that the explicit type cast of the string to an integer uses `convert_to_long` (which works differently than the `is_numeric_string`

How can you tell if a number is even or odd without using any condition or loop?

Hide answer

```
$arr=array("0"=>"Even","1"=>"Odd");

$check=13;

echo "Your number is: ".$arr[$check%2];
```

What is the difference between `include_once()` and `require_once()`, which one would you use in circumstances where you need to connect to a database, and why?

Hide answer

`include_once()` or `include` allows a file to be included, and in cases where the file is missing or has the wrong name, we receive an error message and execution will still continue regardless.

Toptal

What is the difference between a session and cookies?

Hide answer

A session stores the value on the server and cookies store the value in the user's browser.

```
$str = 'drinking giving jogging 喝 喝 passing 制图 giving 跑步 吃';  // Example input
```

Highlight all the Chinese characters in red and return the string.

Hide answer

```
$str = 'drinking giving jogging 喝 喝 passing 制图 giving 跑步 吃';
$string = explode(' ', $str);
$chi = array_filter( explode('_', preg_replace(array('/[^\p{Han}? ]/u', '/(\s)+/'), ar
$value = array ();
foreach ($string as $s) {
           if (in_array($s, $chi)) {
               $value[] = ''.$s.'';
           } else {
               $value[] = $s;
           }
}

return (implode(' ', $value));
```

Write a sample of code showing the nested ternary conditional operator in PHP.

Hide answer

Toptal®

Consider the following code:

```
$str1 = 'yabadabadoo';
$str2 = 'yaba';
if (strpos($str1,$str2)) {
    echo "\"" . $str1 . "\" contains \"" . $str2 . "\"";
} else {
    echo "\"" . $str1 . "\" does not contain \"" . $str2 . "\"";
}
```

The output will be:

```
"yabadabadoo" does not contain "yaba"
```

Why? How can this code be fixed to work correctly?

Hide answer

The problem here is that `strpos()` returns the starting position index of `$str2` in `$str1` (if found), otherwise it returns `false`. So in this example, `strpos()` returns `0` (which is then coerced to `false` when referenced in the `if` statement). That's why the code doesn't work properly.

The correct solution would be to explicitly compare the value returned by `strpos()` to `false` as follows:

```
$str1 = 'yabadabadoo';
$str2 = 'yaba';
if (strpos($str1,$str2) !== false) {
    echo "\"" . $str1 . "\" contains \"" . $str2 . "\"";
} else {
    echo "\"" . $str1 . "\" does not contain \"" . $str2 . "\"";
}
```

Note that we used the `!==` operator, not just the `!=` operator. If we use `!=`, we'll be back to