## Ternary Operator

A way of compacting if-then-else constructions:

condition ? expression1 : expression2;

‘expression1’ is executed if the condition is true. You can omit this true value, creating an ‘Elvis’ operator:

condition ?: expression;

And it can be used like this:

$a = true;

$b = $a ?: ‘foo’;

echo $b; // 1

What I think is happening there is ‘if $a evaluates to true, then make $b equal $a’, or ‘if isset($a) then $b = $a else $b = ‘foo’’. So that makes sense as a way of say using an input value if it exists and having a default value if it doesn’t. No wait: ‘This shortened version of the ternary operator is not suitable for testing if a variable exists, as the interpreter will throw a warning in this case’. Jesus.

## Null Coalescing Operator

*This* is what you should use (as a ternary operator) to test if a variable exists.

$sortDir = $\_GET[‘sort\_dir’] ?? ‘ASC’;

These can be chained:

$sortDir = $\_GET[‘sort\_dir’] ?? $defaultDir ?? ‘ASC’;

So, you can go through a succession of vars before ending with a concrete default value.

Seems like the Elvis operator is just something to be avoided.

A tricky example:

$a = "apples" <=> "bananas";

$b = $a ?? $c ?? 10;

echo $b; // -1

So what is happening here is, if $a is set, make $b equal to $a. $a *is* set, with a value of -1. The temptation is to think that this -1 will be interpreted as condition=False, so echo $b would return 10 ($c not being set).

But if you do this:

$c = 'foo';

$a = "apples" <=> "bananas";

$b = true ?? $c ?? 10;

echo $b;

…you get 1. Same if the condition is $a == -1. I do not understand what is going on here.

$a = "apples" <=> "bananas";

$b = $c ?? 'bar' ?? 10;

echo $b;

This returns ‘bar’.

If the first expression is set, then $b takes the value of that expression.

‘if this var is set, take its value’, which is not the same as ‘if this *condition* is true…’.