Common Mistakes with Conditionals

In this article will go over some mistakes that are easy to make when writing more complex programs with conditionals.

In this article, we're going to walk through some common mistakes programmers make when they start working with conditionals. Knowing about common mistakes doesn't necessarily stop us from making them, but it can help us catch them more quickly!

Mistake 1: Not treating expressions as distinct:

Let's say we want to print something if our variable `$num` is equal to `1` or `2` or `3`. Why doesn't the following code work?

```
$number = 5;
if ($number === 1 || 2 || 3)
{
  echo "Your number is 1, 2, or 3";
} else {
  echo "Your number is NOT 1, 2, or 3";
}
```

The code above prints `"Your number is 1, 2, or 3"` even though our number is `5`...
In a compound condition, each expression is treated separately.
Since `2` and `3` are *truthy* values, the condition above is the equivalent of `FALSE || TRUE || TRUE` which evaluates to `TRUE`. Here's one way we could rewrite the broken code:

```
$number = 5;
if ($number === 1 || $number === 2 || $number === 3)
{
  echo "Your number is 1, 2, or 3";
} else {
  echo "Your number is NOT 1, 2, or 3";
}
```

The computer will treat each expression as separate from the last, so we need to account for that in our code.

Mistake 2: Omitting Parentheses:

When discussing logical operators, we mentioned their *operator precedence*—the order in which the computer will evaluate expressions with multiple operators. The precedence of arithmetic operators may feel a little more comfortable, but remembering that all operators have a precedence in relation to one another can take getting used to...

```
TRUE || FALSE && FALSE; // Evaluates to: TRUE

TRUE || FALSE and FALSE; // Evaluates to: FALSE

$my_bool = TRUE and FALSE; // $my_bool is TRUE
```

This might not crop up that often, but recall that we can avoid any risk by using parentheses to force expressions to evaluate in the order we intend:

```
(TRUE || FALSE) && FALSE; // Evaluates to: FALSE
(TRUE || FALSE) and FALSE; // Evaluates to: FALSE

$my_bool = (TRUE and FALSE); // $my_bool is FALSE
```

Let's look at another example:

```
$withdrawal = 120;
$balance = 110;
if (!$balance < $withdrawal){
     echo "Success";
     $balance -= $withdrawal;
} else {
  echo "Insufficient Funds";
}
```

Even though the `$balance` is less than the `$withdrawal` amount, the code above makes the completes the withdrawal... why? Without parentheses, the computer first evaluate `!$balance` and then checks if that value is less than `$withdrawal`. The expression `!$balance` is converted to the falsy numeric value of `0`, therefore the expression `!$balance < $withdrawal` returns `TRUE`.

When using the `!` operator to negate a logical expression, we should wrap the expression in parentheses.

```
$withdrawal = 120;
$balance = 110;
if (!($balance < $withdrawal)){
     echo "Success";
     $balance -= $withdrawal;
} else {
  echo "Insufficient Funds";
}
```

And check out the following broken code:

```
if ($age < 0) || ($age > 120) {
  echo "That's an invalid age!";
}
```

The code above will cause an error. Did you catch the mistake? We've wrapped our separate conditions in parentheses, but we also need to make sure the entire condition is inside parentheses! We should rewrite that code as:

```
if ( ($age < 0) || ($age > 120) ) {
  echo "That's an invalid age!";
}
```

Mistake 3: Not thinking like a computer:

One of the most difficult things about learning to code, is learning to think the way a computer "thinks". It's important to look at expressions like a computer would. Consider the following code to prompt a user if they enter an invalid response:

```
if ($response !== "yes" || $response !== "no"){
  echo "You must type either yes or no";
}
```

In the code above, we intended to catch situations where the user either didn't enter `"yes"` or `"no"`. We wrote the code the way we might say it, "If the response isn't yes or no..." But the expression `$response !== "yes" || $response !== "no"` will always evaluate to `TRUE` even when the `$response` was actually `"yes"` or `"no"`! If the `$response` was `"no"`, for example, the expression will evaluate as `TRUE || FALSE` which evaluates to `TRUE`.

We can fix this broken code by replacing the `||` operator with the `&&` operator:

```
if ($response !== "yes" && $response !== "no"){
  echo "You must type either yes or no";
}
```

When our code isn't working the way we expect, we should walk through it like a computer—we should "be" the computer and try to read each line without our natural human bias.