Logical Operators

[(https://github.com/learn-co-curriculum/phase-0-pac-1-logical-operators)](https://github.com/learn-co-curriculum/phase-0-pac-1-logical-operators) [(https://github.com/learn-co-curriculum/phase-0-pac-1-logical-operators/issues/new)](https://github.com/learn-co-curriculum/phase-0-pac-1-logical-operators/issues/new)

Learning Goals

- Describe how to use `!` to negate an expression
- Describe how to convert an expression to a Boolean using `!!`
- Define the `&&` and `||` operators
- Describe how to link conditions using the `&&` and `||` operators
- Practice What We Learned

Introduction

In this lesson, we will continue to expand our tool set for creating Boolean expressions by learning about logical operators. Using JavaScript's three logical operators, NOT (`!`), AND (`&&`), and OR (`||`), we'll learn how to negate and combine expressions. These operators, in combination with the equality and relational operators we learned earlier, will enable us to create more complex and sophisticated Boolean expressions.

Describe How to Use `!` to Negate an Expression

`!` NOT

In an earlier lesson, we learned about truthy and falsey values in JavaScript. The logical NOT operator (`!`), also called the *bang operator*, operates on an expression, returning the opposite of the expression's truthiness. If `x` resolves to a truthy value, `!x` returns `false` . If `x` is falsey, `!x` returns `true` . Let's add the following code into our REPL and see what the values are.

```
const truthyValue = 'This value is truthy.';
const falseyValue = 0;
```

```
!truthyValue;
```

Verify the above code works in the REPL console. You will see the bang operator in action, returning the reverse of `truthyValue` 's truthiness. Be sure to verify that it works for `falseyValue` as well. Remember to click the run button to reset the console if you get an error or want to clear out the code.

Describe How to Convert an Expression to a Boolean Using `!!`

In an earlier lesson, we passed values into the `Boolean()` *constructor function* to check their truthiness. We'll learn all about constructor functions later in the course; for now, just think of `Boolean()` as a function that takes in some input, *constructs* a new Boolean from that input, and outputs the newly constructed Boolean.

As a shorter way to convert any value into a Boolean, we can use two NOT operators. Let's try running `!!truthyValue` in our console to see the difference.

The JavaScript engine reads from left to right: it sees the first `!` and looks to the right to check what we're asking it to invert (`!truthyValue`). It then sees the second `!` and looks to the right *again*, this time finding our `truthyValue` variable. At this point, the engine resolves `truthyValue` to `"This value is truthy."` , which (as it tells us) is truthy. It then executes the inner `!` operator on it. `!truthyValue` returns `false` , so instead of `!!truthyValue` JavaScript is now evaluating `!false` . Executing the outer `!` operator on `false` returns `true` .

Try inverting various values in the REPL above to get a feel for the NOT operator. See what happens when you stack a ton of them: `!!!!!!!!truthyValue` .

On to the next!

Define the `&&` and `||` Operators

`&&` (AND)

The logical AND (`&&`) operator takes two expressions:

```
expression1 && expression2;
```

The return value of the `&&` operator is always **one of the two expressions**. If the first expression is falsey, `&&` returns the value of the first expression. If the first expression is truthy, `&&` returns the value of the second expression.

Again, if the first expression is falsey, `&&` returns that value and exits *without ever checking the second expression*:

```
false && "Anything";
// => false

// 4 * 0 returns 0, which is falsey
4 * 0 && "Anything";
// => 0
```

If the first expression is truthy, `&&` then returns whatever the second expression evaluates to:

```
true && false;
// => false

1 + 1 && "Whatever";
// => "Whatever"

"The truthiest of truthy strings" && 9 * 9;
// => 81
```

There are three different ways the `&&` operator can be evaluated:

Left side	Right side	Return value	Truthiness of return value
Falsey	Doesn't matter	Left side	Falsey
Truthy	Falsey	Right side	Falsey
Truthy	Truthy	Right side	Truthy

1. If the left-side expression is falsey, the right-side expression doesn't matter at all. The `&&` operator returns the left side's falsey value and finishes.
2. If the left-side expression is truthy, the `&&` operator returns the right side's value (whether it's truthy or falsey) and finishes.

What this means is that the return value of the expression will be truthy if the values on either side of the `&&` are *both* truthy, and falsey otherwise.

If you're feeling a little confused, that's ok. This is one of those concepts that's a bit hard to understand unless you've played around with it in code. You will have an opportunity to practice at the end of the lesson.

`||` (OR)

The logical OR (`||`) operator also takes two expressions:

```
expression1 || expression2;
```

As with `&&`, the return value of the `||` operator is always **one of the two expressions**. If the first expression is truthy, `||` returns the value of the first expression. If the first expression is falsey, `||` returns the value of the second expression.

If the first expression is truthy, that value is immediately returned and the second expression is never evaluated:

```
true || "Whatever";
// => true

1 + 1 || "Whatever";
// => 2
```

If the first expression is falsey, `||` returns whatever the second expression evaluates to:

```
false || "Whatever";
// => "Whatever"

1 === 2 || 8 * 8;
// => 64

"" || "Not " + "an " + "empty " + "string";
// => "Not an empty string"
```

There are three different ways the `||` operator can be evaluated:

Left side	Right side	Return value	Truthiness of return value
Truthy	Doesn't matter	Left side	Truthy
Falsey	Truthy	Right side	Truthy
Falsey	Falsey	Right side	Falsey

1. If the left-side expression is truthy, the right-side expression doesn't matter at all. The `||` operator returns the left side's truthy value and completes.
2. If the left-side expression is falsey, the `||` operator returns the right side's value (regardless of whether it's truthy or falsey) and completes.

What this means is that the return value of the expression will be truthy if *one or both* of the values on either side of the `||` are truthy, and falsey otherwise.

Practicing What We've Learned

Okay, let's get some practice with logical operators. Take a look at the first un-commented out line in the snippet below: `0 && false;` . What do you think the expression will return? Think it through and come up with an answer, then copy & paste it into the REPL console and hit enter to check whether you're right. When you're done with the first expression, work your way down through the remaining expressions in turn. Be sure you think each example through and figure out your answer before running the code.

If you're having difficulty with the examples, try following this procedure:

1. Evaluate each side of the operator individually: what is the *return value* of each individual expression, and what is the *truthiness* of each of those values?
2. Find the corresponding row in the appropriate table above and refer to the 'Return value' column to determine the return value of the full expression.

Finally, once you're done with the provided expressions, experiment with some examples of your own to cement your understanding.

```
//What should each of the following expressions return?
//Once you have an answer, copy the expression in question into the console and hit enter to see if you're right!

// Practice with AND
0 && false;

0 && true;

true && NaN;
true && !1;

!0 && "This is a string";

!0 && "";

!0 && !!"";

// Practice with OR
0 || false;

true || false;

true || 1;

!true || !false

!1 || !0
```

Conclusion

In the last few lessons, we've been introduced to powerful tools for creating Boolean expressions: comparison operators (equality and relational) and logical operators. With these tools, we can construct very sophisticated expressions. A bit later in the course, we will learn how to use these expressions to execute code conditionally, which will enable us to implement powerful logic in our programs.

Resources

- MDN (https://developer.mozilla.org/en-US/)
- Logical operators (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators)
- Review of conditionals, comparisons, and logical operators (https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/conditionals)