So far in this module, you have learned how to write code that will make your program adapt to the state of variables inside your programs. This is a powerful tool, and one that makes programs much more interesting. In this video, we're gonna introduce three additional operators that are useful for writing compound Boolean expressions.

To do that, let's take a look at example. Suppose that we wanna design an algorithm that will set a variable called days to the correct number of days for a given month. To do this, we refer to the following poem. Thirty days hath September, April, June, and November.

All the rest have 31, excepting February alone. And that has 28 days clear and 29 in each leap year. So what's the input to this problem? Hopefully, you saw that the input for this problem is the month, for which we wanna calculate the number of days. To keep things simple, we're gonna assume that all months are entered as three-letter abbreviations.

In the exercises that follow this video, we're gonna ask you to think about how you would modify this code to handle different ways of expressing the month. Now, to solve this problem, we could go each month one at a time and determine how many days it has. That means we would have January, February, March, April, May, June, July August, September, October, November, December.

The poem says 30 days hath September, April, June, and November. That means that September has 30 days, and April has 30 days, and June has 30 days, and November has 30 days. All the rest have 31 except February. That means January has 31. March has 31. May has 31.

July has 31. August has 31. October has 31. December has 31. That leaves February, which has 28 days, assuming that it's not a leap year. Just based on this list, we could write a long if statement. Boy, boy, would it be a long one. There would be 12 different blocks.

Instead, let's combine the months that have the same number of days, and let's put it into a flow chart. So as always, in my flow chart, I'm gonna start with the start block in the oval. And then, I'm gonna do input. Input, as we said, is the month.

Okay, then we need to know how many days each month has. Let's start with the easy one, February. So if month is equal to February, then the true side, days is equal to 28. Otherwise, that's the false side, we need to know if the month is September, April, June, or November.

So this is gonna be a big decision. We wanna know if the month is equal to September, or if the month is equal to April, or if the month is equal to June, or if the month is equal to November. If it is, then the number of days is equal to 30.

Otherwise, the number of days is equal to 31. That concludes that decision, and this concludes the other one. And then, we'll go and we'll print the number of days before we end. Great. It

would be really nice if we could write this efficiently in Python, particularly this decision as to whether or not the month is September, April, June, or November.

And yes, we can. What we need to do this are logical operators. Logical operators are operators that are specifically designed to express compound Boolean expressions by combining one or more simple Boolean expressions. One of the most common ways to express the value of a compound Boolean expression is through the use of a truth table.

A truth table is a table that shows all the possible combinations of truth values for two or more Boolean expressions, and the values of compound Boolean expressions built from applying logical operators. If that was confusing, don't worry, we'll show you how that works. But for now, we're gonna use them here to help us define Python' s three logical operators.

And, or, and not. Python's and operator is defined as follows. Suppose a and b are two Boolean expression, then a and b is true only if a and b are both true. If either one is false or both are false, the compound expression is false. This is clearly expressed in a truth table.

Notice that we have all the possible combinations for the values a and b. We then apply the and, which is true if and only if both a and b are both true and false otherwise. When evaluating and expressions, if the first expression is false, Python will not bother to evaluate the second part of the expression simply because false and anything will always be false.

This is called short circuit evaluation. Python's or operator is defined as follows. Suppose that a and b are two Boolean expressions. Then, a or b is false, only if a and b are both false. If either one is true or both are true, then the compound expression is true.

Again, we have a situation where we have all the possible combinations of values for a and b. Then, we apply the or, which is false only if both a and b are false, and true otherwise. This is often referred to as inclusive or, and is different from what happens in casual conversation where you might be asked would you like soup or salad?

The implication is that you can have one or the other but not both. That is exclusive or. In computer science, or is inclusive not exclusive. Python will similarly evaluate or expressions using short circuit evaluation. In this case, however, it is looking for the first expression to be true.

Look at the table, and you will find true or anything is always true. Lastly, Python's logical operator not is also known as negation, and is different because it is a unary operator, meaning it has only one operand, instead of a binary operator. It reverses the truth value of the expression to which it is applied, as shown in this truth table.

Negating expressions is often a point where programmers make mistakes, and it can take some careful thought. In mathematics, if x is equal to y is true, then x not equal to y is false. In Python,

the expression not x is equal to y is the same as x not equal y.

Similarly, not x bang equal y is the same as x is equal to y. What about x less than y is false? What's the equivalent to not x less than y? Hopefully, you said that the equivalent to not x less than y is that x is greater than or equal to y.

Did you forget the equal sign? This is a common mistake when negating relational operators. We will see more about negation later in this module. Now, let's go back to our program. Remember the flow chart that we wrote? Okay, we start by inputting the month, and then the easy one is February, it has 28 days.

Otherwise, it's the months September, April, June, or November. Those have 30 days, otherwise there's 31. Now, we know that we can use a logical operator or to write this in Python, because only one of those things should be true, okay? So let's write this in Python. I'm gonna start by defining main like I always do.

And then, the first thing I need to do inside of main is to input the month. So month is equal to input. Enter month, and then I need to determine the number of days. So if month is equal to February, remember our flowchart. Then, the number of days is equal to 28.

elif the month is equal to September, or the month is equal to April, or the month is equal to June, or the month is equal to November, then days is equal to 30. Else Days is equal to 31. And then, I'll print that the month has days, days.

Just so that we can print it. There's only three possible paths. Much better than 12. The other operators can be used similarly, and we will practice doing this in some of the exercises at the end of this video and in the assignments. Like all operators, logical operators have precedence and associativity that can be added to the precedence table that we've been building.

Notice that logical operators have a lower precedence than the comparison operators, and that will come in handy later during this course. Okay, now we've covered the basics of conditional statements from learning how to write both simple and compound Boolean expressions, to learning how to use them in conditional statements to change the behavior of our program.

Let's make sure to take some time to practice them by completing the exercises that come after this video. That's all for now. Thanks for watching Align online.