# Transcript: Expressions

>> Now let's turn our attention to expressions. An expression is made up of one or more variables, operators, and or method invocations and this all evaluates to a single value. You see them mostly on the right hand side of an assignment statement, but they can be lots of places.

Arithmetic expressions actually, computing numeric results. And, and they use the numeric operators you're familiar with. Addition, subtraction, multiplication, division, and remainder or modulus. You may not be so familiar with it. Do note that multiplication uses an asterisk to represent multiplication. So that's, when you wanna multiply a times b would be a*b.

Now, if either operand is floating point, in our case mostly double, then the result is gonna be a floating point value. So let's take a look at division and remainder with respect to integer types. In Java, we do integer division, and, and when we do that the fractional part is discarded.

So if we've got one of our arithmetic operators, well, let's say division. And both operands on either side are ints. Then we're gonna do integer division and lose that fractional part. For example, if we had 14 / 3. That would end up being 4, and we would throw away the fractional part, which would be the remainder.

Think of long division back when you do division where you end up with a quotient and a remainder. So this is the quotient. If we had 8 / 12 that would be 0. 12 doesn't go into 8. There would be a remainder there. If we wanted to get at the remainder, we would use the mod or remainder operator.

And in this case, here we have 14 remainder 3, and, and the way we would really say this is what is the remainder of 14 / 3? And of course that would be 2. And the other example there, 8 / 12, would give us remainder of 8. Recall it's 0 and the quotient is 0 and the remainder is 8.

So let's look at a, a little example that shows us some of these remainders. In fact, the ones we just did are here, and, and some others. So here's 14/ 3, we want the remainder of that. And then here we've got a -14. And then on this next one here on line 14, we've got a positive 14 / -3, and then on 15 -14, and we've got a -3.

And then we throw in some real numbers in there, we can actually use the remainder operator on those as well. So, let's run this in debug mode. And begin stepping. So there's the one we know about, 14/ 3, the remainder is 2. This one, is not as obvious, but there we get a -2.

And then here we get a positive 2. If you'll work this out in long division, you'll see that even though, the divisor is a negative number, the remainder is gonna be positive. Same thing here. You might think if you've got two negative numbers you're gonna end up with a positive result.

But the reality of it is, you end up with a -2. The rule is that, whatever the dividend sign is, in this case, the first number, if it's negative, then the remainder's actually gonna be negative. It doesn't matter if the devisor is negative. Then here's an example where we 6 / 1.5, and that remainder is 0.

Here we've got a a fractional remainder. And, again, this is because these are, are floating point numbers rather than integers. It's the integer division where we'll use remainder most often. So one more step and this program ends. So let's revisit assignment. Must keep this in mind here, we've got a left and right hand side.

And sometimes they can contain the same value. On the left hand side, we can only have one variable, but in this case, we've got count = count + 1. Now this says, this really is an assignment statement, because in mathematics, count could never be equal to count + 1.

Remember you could take count away from both sides and then we would have 0 = 1, which is not true of course. So in this case here, we evaluate the right side. Whatever that expression yields and then is assigned over to the left overwriting the, the original value.

Now, in, in programming we do a lot of incrementing and decrementing, as in adding one and subtracting one. And so we have a sort of a shortcut operator here of ++ and ++ is increment, -- is, is decrement. So for example, we could have count ++; making this a statement, not an expression.

And this is functionally equivalent to saying count = count + 1 and seems it's just really a short-hand notation for this and, and it's used, used frequently and we'll use it as well. Now we could also do this with other pairs of operators if you will and this is called assignment operators.

For example, we could have num += count and note that the + here is a binary and so this really says num + count. And then whatever the result is, we wanna assign it back to num. So this is equivalent to num = num + count. Have to think about this a little bit, but in general the += just means we take the two operands, num + count and then the result we assign back to num.

Again, just a shorthand notation. So looking at characters, if if we wanted to assign them, here's a char tropGrade = A. Next example, char terminator = ':', separator = Here we did two assignments on one line separated by a comma. Now, keep in mind that a a primitive character variable only holds one character, while a string object holds multiple characters.

The difference would be that, you'd, you'd have double quotes, and you'd be assigning it to a string, which is an object, rather than a primitive type, and we'll talk more about that, later. Next we've got boolean. A boolean value represents true or false. Those are the only values it can take on.

So we've got the reserved word true and false. Again, the only valid thing, here's an example. Boolean done = false. Boolean variable, you can use a boolean variable, it would be appropriate, if you want to keep up with any two states, such as a light bulb being on or off, yes or no, and and so on.

So you can keep up with these states, but remember the value is actually gonna be true or false. Now, what we use with boolean operators, or operands if we had boolean operands. We can use the relational operators, and here they are. So for an arithmetic expression, we use the arithmetic operators plus, minus, divide, and so on.

But for boolean expressions, we're gonna use these relational operators if you will. So the first one here is equal and notice the operator we use is ==. This is to show that it's different from assignment. Single equals would mean assign but == means we wanna know if two things are equal.

This second one, !=, is not equal. You might thing that, think this means really equal, but, no, it means not equal. And then the last four you're probably familiar with. Less than, less than or equal, greater than, greater than or equal. We use these with boolean types. So here's an example of a boolean expression 89 > 50.

Now that's either true or false. And, and we know this must evaluate to true or false because we're assigning it to a boolean variable. So 89 is greater than 50, that is true. And so that would set set true here to to the the variable greater. Here I've got int temp = 99, and then boolean isCold = temp < 50.

And the same sort of thing in this case, temp is 99. We just assigned it on the line above, and so 99 < 50 that's gonna be false. And so the variable isCold would be set to false. Clearly whoever wrote this was from the south if they thought that 50 was cold.