Welcome to this video on using if statements in Python. In the video Introduction to Programming we were executing either single statements or several statements that are executed in sequence, top to bottom. Sometimes, however, our program needs to execute one branch out of several potential options. That is, our program needs to make a selection. There are three options for controlling how our programs execute. The first is sequence, which is the top to bottom execution that we've seen so far. The second is selection, as just mentioned. The third is repetition, where the same thing is executed over and over. We'll get into that in a later video. This video will focus on the selection control structure using the if statement. An if statement is a way to define optional paths in our program with the decision as to which path to take dependent upon conditions. If statements can take three forms, the first is the if statement by itself. For example let's say that we make a variable called blue and make it a boolean with a value true. Notice the capital T in true, that's important. We can now use this variable as the condition for our if statement. We could write if blue, on next line, print 'colour is blue'. Let's delve into what's happening in this statement a little more. Our blue variable was assigned the value true, then we used it in the if statement. We would read our if condition as 'if blue is true then execute the print statement'. As our blue variable is true, the print statement executes and our string prints to the screen. Now you try setting blue to false and see what happens.

Before we move on, do you notice how the code inside the if statement is indented? In Python this indentation is used to demarcate levels of code. The indented code under the if statement lives inside the if structure only. If it was indented back in line with the if statement, it would be part of the larger program, not the if statement. Try playing with the indentation to see what happens. This very simple type of if statement has only one branch. It either does something when true or the program skips over it and goes on to whatever is next in its top-to-bottom execution. What if we want to have two possible paths? Then we can use something in conjunction with our if, called an else statement. If we add an else to our blue example we would write else on the next line, then have a

different print statement saying that the colour is not blue. Now the if-else combination has a path for both a true and false condition. There is one more type of if control structure and that's used when we have three or more branches. To give an example of this however we need a condition that has more than two options. Our blue boolean variable has only two options, true or false.
So let's first have a look at some other ways we can build if conditions before we come back to the final type of if structure. The way an if condition works is that if the expression evaluates to true then the if executes. If not, the program progresses without executing the code inside. This is exactly what happened with our blue variable because the variable itself was already boolean. With data types other than booleans we can achieve the same thing by using special operators that allow us to make comparisons. Here are six of the most common comparison operators. These are usually placed between two values to compare the value on the left to the value on the right. Let's take a look at some examples in the IDLE shell. Let's make two variables, X with the value of five, and Y with a value of one. Then we'll use the less than comparison operator to ask the question, is X less than Y. As you can see this evaluated to false as X with a value of 5 is larger than Y. Try replacing the less than arrow with the greater than arrow and see what output you get. Notice that we're getting a boolean output from this comparison and we know from our blue example that a boolean is what we need for if statements. Okay,
now we can go back to our final if structure. We're going to work with two integer variables, num1 will have a value of 3, and num2 will have a value of 5. What we want to know is if num1 is larger or smaller than num2.
First let's create the first if which will test if it is larger. So we write if num1 is greater than num2, print num1 is larger. Then our next statement we use elif, which is read as else if. This allows us to have another condition.
Unlike before when we just used else we will write else if num1 is less than num2, print num1 is smaller. That's two conditions which test whether num1 is larger or smaller. Can you guess what our final statement tests? The one possibility that we are missing is if num1 is the same, or equal

to num2. As that is the last and only other option we can write else print num1 and num2 are the same. Try changing the values of num1 and num2 to force each of these paths to execute. So far we've only dealt with single part conditions but what if we need to be able to handle a two part condition. For this example let's work with hunger and time of day. We're going to have three meal times 9 a.m.for breakfast, 1:00 p.m. for lunch. and 7:00 p.m. for dinner. Let's make a variable to hold the time. We'll call it time and assign it a value of 19 to represent the 24-hour clock
make a variable named amhungry and assign it the boolean value of true.
Now we need to write our if statements which will be very similar to before as we'll need three branches, one for each meal time.
Let's code those three just focusing on the time for now. So we write if time equals nine, print have breakfast, else if time equals thirteen, print have lunch, else, print have dinner. Great, that should work.
But we haven't taken into account whether or not we're hungry. We only want to eat if we're hungry. Think about how I just said those sentences. We only want to eat if we are hungry, so it must be true that the time equals one of the meal times and we must be hungry. The way we code this is very similar to how we say it. Let's add the amhungry boolean. To the first if we add and amhungry and that's it. Now this condition reads, if time is equal to nine and amhungry is true, the expression is evaluated left to right and due to the 'and' both parts must be true in order for the if to execute. We can add the same to the else if. The else, however, is a problem as it would execute when the if and else if were false, regardless of whether or not it's seven and we're hungry. We really need this to be another else if, so let's make that change. As you can see we don't necessarily need an else if we don't have to have a catch-all that will execute if all other conditions are false. Again
try modifying the values of time and amhungry to see how these different paths can be triggered. There's one thing that we're still missing. Regardless of the time and whether or not I'm hungry if the food option is cake or chocolate I'm eating it. We currently have no way to deal with this kind of statement. We want

to be able to eat if the option is cake or chocolate ignoring the time or current level of hunger. First let's create another variable called option that will hold the type of food on offer. Let's assign it the value 'cake' as a string. Now we need another condition in our if structure. Let's create another elif at the bottom that reads, else if option equals cake, print I always eat cake or chocolate. This would work if the option is cake but the condition would be false if option were assigned the string chocolate, so we add the second part. We write or option equals chocolate, now it will catch both. What's really important to understand is that the conditions are evaluated top to bottom so if the time wasn't a meal time and we weren't hungry each of those first three conditions would be false but if the option was cake or chocolate the final else if would be true. If time was one of the meal times and we were hungry it wouldn't matter whether or not the option was cake or chocolate because one of the first three conditions would be true. Essentially this would mean that we'd eat cake or chocolate as a meal which is totally fine. One other comparison operator that we need to understand is 'not'. This simply negates the evaluation of an expression. So using our blue example we could write if not blue, which would it basically mean if it's any colour other than blue, do something. Let's write an example that uses a number of the concepts we've learned in this video and the Python standard library video. In this exercise we want to generate a random number between 1 and 10. Then we want to give a user of our program an option to guess the number. If the guess is too low we'll print that it's too low and the same if it's too high. If it's the correct guess we'll congratulate them. Think about this problem and see if you can think of some components and control structures that we'll need. We'll need a variable to hold the number, so let's make that first. We'll call it number and assign it a value that's a random number between 1 and 10. To get that, we need a function called randint that's not part of the Python standard library. We can get it by importing random at the top of our program, so number is assigned the value of our call to the random function. That's our random number sorted. We want to allow a user to

guess it which means we'll need a way to store their guess, another variable. We make a variable that we'll name guess that will be assigned the value of a call to the input function with the prompt 'Please enter your guess'. Remember that we want to get a number from the user. Can you recall what type of variable is returned by the input function? It's a string, so we need to convert it to an int. We use the int function and wrap the entire input call.
Now we just need those options that tell the user whether or not the guess was correct. The first if can test whether the guess is too low, or less than the number, the second else if condition, can test if the guess is too high, or greater than the number. The final else condition is the only remaining option, a correct guess.
Hopefully you'll see how you can create if conditions to solve problems where there are multiple possible outcomes. As always practice makes perfect.
See you in the next video.