Python – Code Control Structures

##### Speaker – Curt Knowles

##### [00:00:00.860]

Hello, everyone. In this video we're going to continue to build on our understanding of the Python programming language by looking at a few different code control statements and how you would write those in the Python language. So first of all, we're going to have a short discussion on understanding how boolean expressions, these are true/false expressions, are evaluated. We have to have that as a precursor to looking at conditional constructs such as if statements, while statements are for statements, because those statements use boolean expressions to determine the course of action and the path that your code is going to take in your program. So looking at boolean expressions.

##### [00:00:51.440]

As I said, these are expressions that can be evaluated to either true, which is a 1 value or false, which is a 0 value. There are a combination of relational and logical operators that are used to evaluate Boolean expressions. Some of the relational operators are the equal and not equal to, the greater or less than, the greater than or equal to, less than or equal to. And I want you to note the "equal to" is a == sign in Python. This differs from the =.

##### [00:01:30.400]

The = is used for assignments. So if you're assigning a value to a variable, that would be the =, but the == is used for doing comparisons. So if you say "if A equals to B", you would use the ==, not the =. Also the not equal to, instead of being a <>, that is what we call a bang or exclamation mark equal (!=).

##### [00:01:59.820]

That is the operator to evaluate "not equal to" expressions. Also, you use the logical operators in evaluating boolean expressions that we learned about during our second lesson in the course a couple of weeks back using the AND, OR, and NOT truth table operations to evaluate those expressions. These do have an order of precedence. If you use more than one of them in a single line, any NOT would get evaluated first, and then any AND and then any OR would be or would be evaluated last.

##### [00:02:36.630]

Here are some examples of some basic relational operations, boolean expressions.

##### [00:02:43.280]

The way you can read these for the one that is highlighted. "If age is equal to five, then do something" or "if first\_name is equal to the string 'John'". Going on, we use the != here. "If quantity is not equal to 0", and so on, using the greater than, the less than, the greater than or equal to, the less than or equal to, and then finally the greater than or equal to. And I want to point out to you here that, as illustrated on this slide,

##### [00:03:17.370]

sometimes you can be comparing two variables to each other, in which case you're evaluating the values that those variables contain, or you can be evaluating a variable against a literal value - a numeric literal or a string literal. You can also do math operations on one side of the boolean operation, as in the last example that's now highlighted, where we ask the question "is rate divided by 100 greater than or equal to 0.1?" Some examples of using the logical operators. The way you would read the one that is highlighted would be "is age greater than or equal to 65 and is the city equal to Chicago?" If so, then do a certain set of actions.

##### [00:04:15.700]

The next one, "if city is equal to Greenville or the age is greater than equal to 65, then do this" or "if age is not greater than or equal to 65". And you can go on and you can see how we can combine multiple operators. Here we have two AND operators, two OR operators, and AND and OR operators on the last two. And notice how the parentheses will change the sequence in which the operations are evaluated. So in the first one, the AND will be evaluated first because it's in the parentheses.

##### [00:05:00.660]

However, in the last one on the page, the OR will be evaluated first, since it is now in the parentheses.

##### [00:05:11.540]

Also, you can do string comparisons in boolean expressions, and we need to note how strings are evaluated in Python. Strings are just a set of characters. So strings are evaluated one character at a time, moving left to right, but they are evaluated each character according to a sort sequence order. So digits from 0 to 9 < uppercase letters from A to Z < lower case letters from a to z. So we have a few examples at the bottom.

##### [00:05:54.270]

First, we are comparing "apple" asking "is this less than Apple?" So we're going to start at the left, and the first character we get is a comparison of "a" and "A". These are not the same because Python is case-sensitive. So in our sort sequence is "a" < "A"? No, it's not.

##### [00:06:19.400]

Uppercase letters are < lowercase letters. So this is asking "is apple < Apple?" No, it is not. It is, that's false. It's actually > Apple.

##### [00:06:35.180]

In the second statement, we are looking at the word "App". Is it less than the word "Apple"? And in this case we evaluate the first three characters, the "A", the "p", and the "p". Those are all equal. So we get to the fourth character, which is the "l" and in "App"

##### [00:06:58.350]

there's nothing to compare it to. We're at the end of the string. So what happens here is that Python looks at a blank character and compares it to the "l". Is the blank less than the "l"? Yes, it is.

##### [00:07:17.100]

The blank actually comes before even the digits in the short sequence. So the blank is less than "l". So we stop the comparison at that point, and this statement is true. In the third statement, we compare the string "1" to the string "5", so we compare these as characters and the character "1" will be, they're in the same sort sequence, which is the digits.

##### [00:07:46.240]

So it goes from "0" to "9", so the "1" will be less than the string "5". In the last one, we compare the string "10" to the string "5", and again we do character by character. So as soon as we evaluate the first one is "1" less than "5"? That is also true. Okay, now we have looked at Boolean expressions.

##### [00:08:15.450]

We can go on to the if statement, the while loop and the for loop, so you can see the basic syntax of the if statement. Here a couple of things that I will point out to you. Note the colons at the end of the if statement and all of the other clauses that are part of the statement. We'll tell you what these are in the next slide. Notice the indentation and the fact that you can have one or more statements under each separate clause of the if statement.

##### [00:08:48.480]

So depending on the logic of one or more conditions, the outer statements being the conditions, multiple paths action may be taken by your program with an if statement. So here's the process of how the statement works.

##### [00:09:06.300]

First of all, an if statement will always contain an "if" clause. In addition, it can optionally contain one or more "else if" clauses, which the keyword for that Python is "elif", and it can optionally contain one and only one "else" clause. It cannot contain multiple "else" clauses. When the if statement is first executed, the "if" clause gets evaluated first. If the boolean expression in the "if" clause is true, then the statements that are under the "if" clause get executed.

##### [00:09:45.490]

Once those statements get executed, the entire if statement ends, no other clauses are evaluated. If the boolean expression in the if statement is false, then the first "else if" clause, if it is there, will be evaluated. If the boolean expression in the first "elif" clause is true, then the statements in that clause are executed, and the if statement in general will end after those statements get executed. Otherwise, if the boolean statement in the first "elif" is false, then the next "elif" clause will get evaluated. That will continue until a condition in one of the "elif" clauses is true.

##### [00:10:30.440]

If you get to the end of all of the "elif" clauses and nothing has yet evaluated to true, then if there is an "else" clause, it will automatically be executed. The statements under the "else" clause would automatically be executed at that point.

##### [00:10:52.060]

Notice that with this process, logical workflow process, only one block of statements in the entire statement are going to be run each time an if statement is executed, because once a clause is evaluated to be true the statements under that clause, wherever it is, get executed and then it drops out of the entire statement from there, not evaluating any of the other "if", "elif", or "else" clauses in that statement. Here are a few examples. First of a simple if clause with no "elifs" and no "else".

##### [00:11:30.560]

This is saying "if the age of the person is greater than or equal to 18, then print 'You may vote'".

##### [00:11:40.150]

Here's an example of a basic if-else statement. "If the person's age is greater than or equal to 18, print 'You may vote' otherwise if it is not greater than equal 18 print 'You are too young to vote'". Here's a more involved example where we are looking at invoice\_total and based on the amount of the invoice\_total, assigning a different discount percentage. This contains one "if" clause, two "elif" clauses and an "else" clause.

##### [00:12:13.440]

So we're saying here if the invoice\_total is greater than or equal to 500, then award a discount percent of .2 or 20%. Else if the invoice\_total is greater than or equal to 250, award a discount percentage of 10%. Else if the invoice\_total is greater than 0, award a discount percentage of 0. And if none of that is true, that must mean that the invoice\_total is negative. So print the message "Invoice total must be greater than zero". So notice the mutual exclusivity of these clauses.

##### [00:12:55.860]

Here I'm looking for any invoice\_total greater than equal to 500. If the invoice total is greater than equal to 500, I execute this statement and drop out of the "if" clause - the second "elif" never gets evaluated, the third "elif" never gets evaluated, the "else" never gets evaluated in that case. So if I get to this statement and I look for greater than or equal to 250, I already know it is not greater than or equal to 500.

##### [00:13:24.780]

So really what I'm evaluating here is the invoice\_total between 250 and 499, because anything over 499 would have been caught by the first "if" statement.

##### [00:13:40.080]

Here's another example showing that mutual exclusivity that I just talked about where each clause precludes the other. So here we get a test score from a user, and we look to see if it's greater than equal to 90. If it is, print "A". Otherwise if it's not, that must mean it's less than 90. So next check the 80s. If it's greater than equal to 80, then give them a "B".

##### [00:14:08.660]

Otherwise that must mean it's less than 80. So let's check the 70s and give a "C" if it's there. And likewise for the 60s, and likewise for anything else below 60. So that's the basic "if" statement construct in Python. Now let's go to the while loop.

##### [00:14:30.900]

So the "if" statement is called a conditional structure, because based on decisions or conditions, it'll go in multiple paths. But the while loop is what's called a repetitive structure because it will repeat a series of actions and perform those in a looping fashion until some predefined boolean expression condition becomes false. So as long as the boolean expression is true, the statements in the while loop will execute again. Notice the colon at the end of the while clause, and notice the indentation of four spaces for all statements that fall inside the while loop.

##### [00:15:11.640]

So with the while statement, the process goes that Python begins by testing the condition defined by the boolean expression at the popular statement or at the top of the loop.

##### [00:15:23.800]

If that condition is true, Python will execute all the statements in the while loop, and will continue doing that until the condition at the top of the while loop becomes false. If the condition initially evaluates defaults the first time it's evaluated, then that's a false condition, and Python in this case, would never enter and execute the code in the loop. If the condition never evaluates to false, you can sometimes find yourself in a situation called an infinite loop. And so when you find yourself in an infinite loop, then the program will continue to execute and never end, and so you have to make some correction inside of your while loop at that point

##### [00:16:17.850]

to fix the infinite loop. A while loop uses a boolean condition to determine the number of times that it will execute.

##### [00:16:29.720]

So a couple of examples. First of all, a while loop that continues as long as the user enters "y" or "Y". So we establish a variable called "choice". We're going to initially set it to "y" so that it will enter the loop a first time. Our while statement is "while the choice is 'y' or 'Y'"

##### [00:16:59.470]

then enter the loop. So the first time through, we are going to enter the leap and we're going to print "Hello", and then we're going to ask the user, "Do you want us to say Hello again" (y/n)", and we will put that value back into the variable "choice". Now, once this is the last statement in the loop, so it will pass control back up to the while statement, and it will evaluate "choice" again based on the user's input here. So if the user entered "y" again, we're going to go back in and say "Hello" again and ask them again, "do you want us to say that again?"

##### [00:17:41.790]

Eventually they're going to enter a no here or an "n".

##### [00:17:45.440]

When we loop back up to the while loop, "choice" will then be "n", and this statement will become false at that point. So it will drop out of the while loop then and go to the next statement after the while loop. Notice the indenting. The print("Bye!") statement has the same indenting as the while clause itself, so that is not a part of the while clause.

##### [00:18:11.640]

That's the next statement to be executed once you have completed the while clause. So in this case the user entered "y", we printed it again, then the second time through they entered "n", and we skipped out and said "Bye!". Another example of the while loop here we are going to print the numbers 0 to 4 to the console. So the way we're going to do that in a loop is we're going to use a "counter" variable to print with. So we set the "counter" variable initially to 0.

##### [00:18:46.650]

That's the first number we want to print.

##### [00:18:48.440]

Now we're going to say "while counter < 5". So the first time through the loop counter is 0. We're going to print a 0, and then we're going to set the counter to 0 + 1. So counter now has the value of 1. We go back to the while loop counter - is 1 < 5? Yes.

##### [00:19:13.160]

Going to go in again and print 1, increment the counter to 2. Go back to the while loop - is 2 < 5? Yes.

##### [00:19:22.500]

Print 2, increment counter to 3. If 3 < 5? Yes. Print 3, increment counter to 4. Is 4 < 5? Yes. Print 4, increment the counter to 5. Is 5 < 5? No, it is not. That's false.

##### [00:19:47.000]

So we drop out of the while loop at that point and print "The loop has ended". And here you can see the result of doing all of these print statements.

##### [00:19:58.980]

Okay. The last construct we're going to look at in this lecture is the for statement. So the syntax of the for statement is that you have your "for" clause. We're going to explain what the "range\_function" is. Notice the colon, and again, the indentation with one or more statements on the "for" clause.

##### [00:20:20.200]

Now the "for" clause performs a series of actions that will continue to be performed in a looping fashion until some predefined number of leaps are completed. So the one difference between the "while" and the "for" clause is you use a "while" clause when you're not sure going in how many times that "while" loop is going to need to execute. If you know exactly and it's predefined how many times the loop needs to execute before you go in the first time, then you can use a "for" loop.

##### [00:20:52.940]

So the process of the "for" statement is that it executes the for loop once for each integer in a predefined collection of integers. Each time through the loop, the Interger variable at the top of the loop will receive the next integer in the collection, and then the loop will end after it executes for the last integer in the collection. This predefined collection of integers is called the "range".

##### [00:21:19.280]

So with the "range" you use the "range" function, you can assign one argument to the "range" function. When you assign only one argument that represents where I want to stop executing the loop. Or you can also execute the range function with up to three arguments. A "start" - where I want to start the loop out, where I want to "stop" the loop at, and how much do I want to "step" each time I go through the loop. So here are some examples. So if I say range(5), what that is going to do is going to execute the loop.

##### [00:22:03.630]

It always starts at 0. Unless you specify a different "start" argument, the "start" argument defaults to 0. So range(5) is going to execute through the loop for 0, 1, 2, 3 and 4. Notice that it does not execute through the loop for 5. So the loop is not entered for the "stop" number of the range.

##### [00:22:31.070]

When the for loop hits the "stop" number, then it stops executing the loop right then, it doesn't go in again. So whenever you have a range, the actual stop that will be executed will be one less than the number you indicate for the "stop" argument. So range(1:6). This time we're going to start at 1 and we're going to go up to one less than 6, which is 5. Next, range(2:10:2) - start at 2, end at 10, step by 2.

##### [00:23:05.400]

This is going to start at 2. It's going to step by 2 each time through, so next will be 4, 6, and eight. The next thing through will be 10. So we will not include that because that's our "stop" number. You can also go in the reverse.

##### [00:23:22.370]

So you can go from 5 down to 0 step by -1. So this would step 5 to 4 to 3 to 2 to 1, and that would not include 0 because it's the "stop" argument. So here's a couple of examples. A for loop first that prints the numbers 0 to 4. So we have a "counter" that we're going to use to indicate "this is the number I'm currently at in the range through this iteration of the loop".

##### [00:23:53.940]

So "for counter in range(5)", remember range(5) - what numbers of "counter" is it going to execute the loop for? That's going to start at 0 and it's going to go up to 4. So this loop is going to execute for 0,1,2,3 and 4. Each time it's going to print out the counter 0 1 2 3 and 4, and then once it hits 5, it will drop out of the for loop and print "The loop is ended" and you can see that output here.

##### [00:24:27.280]

Here's another one that prints, sums the numbers from 1 to 4 and it will print that sum. So we're going to set up a variable that is going to collect the sum of our numbers as we go through the for loop. We're going to set up a for loop with a "counter" variable in the range(1:5) because we're adding 1 to 4 = 1,2,3,4 need to be added. So this range will start at 1 and will go and enter the loop for 1, 2, 3, and 4 and will drop out at 5.

##### [00:25:04.040]

So each time through the loop, we're going to take what is currently in sum\_of\_numbers, and we're going to add to that the value of the current counter iteration through the loop.

##### [00:25:13.700]

So I set up a table here to take us through each iteration of the loop and show you the values for each of these variables. So when we go into the loop the first time our "counter" is one, our starting sum\_of\_numbers is 0. That's what we initialized it to in the first statement of the program. So this statement where we do the addition adds sum\_of\_numbers and counter. So it will add those two values and we'll put the result back into sum\_of\_numbers for the next iteration of the loop.

##### [00:25:51.360]

So on the next iteration of the loop, sum\_of\_numbers is now 1. Our counter has been incremented to 2, so 1 + 2 = 3. And we go on like this. The 3 goes over to our next, this should be 3. It should not be 2.

##### [00:26:09.980]

Sorry about that. I will change that on the lecture slides. The starting sum\_of\_numbers is 3. The counter is 3, so 3 + 3 = 6. And then the next iteration, the 6 will come down as the starting sum\_of\_numbers. The counter now has been incremented to 4 and 6 + 4 = 10.

##### [00:26:30.960]

Now the counter gets incremented to 5. So we do not enter the loop for 5. We exit and that's when we go to our print(sum\_of\_numbers) and we will print what is our current sum\_of\_numbers, which is 10. And you can see the output here. Okay.

##### [00:26:52.240]

This has been a brief introduction to the if statement, the while loop and the for loop in Python. If you have any questions for me, please let me know. Send me an email and I'll respond as soon as possible. Thank you very much.