In programming, situations may arise where we need to execute a block of code several times in order to arrive at a solution. A loop allows us to do just that, by providing keywords that can execute a statement or a group of statements multiple times. In Python, there are two types of loops: For Loops, and While Loops. In this video, we are going to cover the basics of the For Loop, how it works, and it’s applications.

A For Loop executes a sequence of statements a specified number of times. In Python 3, a For Loop can be used to iterate through a sequence of values in a list, range, or collection. To create a For Loop, you can use the For keyword, followed by an iterator variable, followed by the In keyword, finally followed by the sequence of elements to be iterated through. A simple example can go like so:

for x in range(5):

print x

Let’s analyze this. From this single For Loop statement, we have been able to print a total of 5 lines even though we’ve only written one print statement. Now let’s break down the For Loop. Our For Loop is defined by the keyword For, which we use to start the statement. Next, we add an iterator, which in this case is the variable x. The In keyword follows this, meaning that we want the variable x to take the form of the elements in our collection. In this case, our collection is the function range() with a passed parameter of 5, which returns a range starting from 0 to 4. Now that we have our For Loop defined, we add a print statement in the block of our For Loop. This print statement executes as many times as our variable x iterates through the elements in range(5). So, since the elements 0, 1, 2, 3, and 4 exist in our range(5), our For Loop will loop for a total of 5 iterations, one iteration for each element in our range. Any statements that we put in our For Loop will be executed 5 times in sequential order. As for the contents of our print statement, our print will display the current value of x at each iteration. That is why it prints 0, 1, 2, etc. on different lines, since our print is being called for a total of 5 times over the course of our loop. Moreover, if we were to make a small modification to our code like so:

for x in range(5):

print x

print(“Hello”)

We can see that our variable x is printed, followed by the string “Hello,” and then the next value of x, and so on and so forth. Remember, the For Loop still executes statements in sequential order.

Let’s look at another example, this time to iterate through a list of strings and determine how many names in the list start with the letter J:

jCount = 0

names = ['John', 'Joe', 'Mary', 'Tim', 'Jill']

for x in names:

if x[:1] == 'J':

jCount += 1

print("In list",names)

print(jCount, "names start with the letter J")

So like before, this For Loop iterates through all the elements in our collection, this time being a list of names. As such, the value x takes the form of each individual element upon each iteration, so on the first iteration, x equals ‘John’, then on the second iteration, x equals ‘Joe’, and so on. Once execution enters the loop, the if statement checks the first letter of our element x. If the first letter turns out to be ‘J’, then we increment our jCount variable by 1. The loop will not include our two print statements at the end because we have not written them in it. This is denoted by the lack of indentation, meaning that these print statements will be run only once each. Once we have gone through all the elements in our list, our loop then exits, and our print statements can be executed. The jCount variable is displayed as 3 in our final print statement as our solution.

Like If statements, For Loops can be nested within one another to perform deeper iterations. This is especially useful when attempting to iterate through a list containing smaller lists or tuples.

myString = [('John', 88), ('Joe', 69),('Mary', 88),('Tim', 70),('Jill', 75)]

Like before, let’s run a For Loop and print the contents of our list.

for t in myString:

print(t)

As expected, our list of tuples if printed. Using a nested For Loop, we can further iterate through the contents of our tuples.

for t in myString:

for x in t:

print(x)

Let’s analyze our output. Each element of our tuples was printed on separate lines, meaning that our print statement was executed for a total of 10 times. This amount is no coincidence, since each tuple contains 2 values, and we have 5 tuples all together. On our first iteration, our t variable became equal to the tuple (‘John’,88). Then, our nested For Loop starts immediately on the next line, where x iterates through the contents of t. x becomes equal to the first value in t, which is ‘john’, and then x is printed. On the next iteration, x becomes equal to 88, and likewise that value is also printed. Since there are only two values in our tuple, our nested For Loop has reached the end of its iteration, and it returns back to the parent For Loop. Since there are no further instructions in our For Loop, t iterates to the next value in our list, which is the tuple (‘Joe’,69). And like before, execution then goes into our nested For Loop again, where the values ‘Joe’ and 69 will be printed, separately, over the course of two more iterations, just like it did for our first tuple. Execution returns to our parent For Loop and t iterates to the next tuple, (‘Mary’, 88), and the cycle will continue to follow this pattern until every element in our list has been analyzed. This process of using nested For Loops essentially allows us to extract all of the information held within our list, even though our list contains a collection of tuples.

One final property about For Loops (and other kinds of loops for that matter) is that they may be escaped at any time with the use of a break command. If the break command is encountered by the loop, then execution will exit the loop regardless of how many more iterations are left. Let’s reuse our earlier example with a modification:

for t in myString:

if t[0][:1] == 'M':

break

print(t)

In this block, we’re checking the first character of our list of names and seeing if it equals ‘M’. If there is a name that starts with ‘M’, then our For Loop will break out of its iteration, regardless of whether or not it is on its last iteration. As we can see from the output, only the first two tuples were printed, as the third tuple contains a name with the letter ‘M’, and so the iteration was broken out of. Notice that the last two names in our list weren’t printed even though they don’t contain names that start with the ltter ‘M’. This is because the moment our if statement caught a name with the letter ‘M’, the break statement forced execution to break out of the For Loop, and so the remaining elements in our list didn’t even get a chance to be analyzed. Thus, our print statement was only executed a total of 2 times in this instance, since our loop was cancelled the moment our break statement was encountered on the third iteration.

To summarize, For Loops allow blocks of code to be executed multiple times in sequence. They require a collection or a range object, as well as an iterator to step through each element. The iterator variable becomes assigned to each value in our collection sequentially, and can be manipulated just like any regular variable in our program. For Loops will continue to run until all elements in a list have been analyzed, or the break command is used. Nested For Loops are essentially For Loops within For Loops, and can be used for deeper iteration if our collection has a second level of ranges (for example, a list of tuples). Nested For Loops are no different from single For Loops in terms of functionality.