In this tutorial, we will test a program using the debugger features in IDLE. This tutorial assumes that you already have IDLE and its required packages previously installed. To start, open IDLE on your computer using the shortcut created upon installation, or if you are using Windows, you can simply open the search menu and open up the program.

We will be testing a sample Python script, fooCount, that will be distributed with this video. Or you can simply copy the program yourself as we go along in the tutorial. First, let’s open up the script, fooCount.py in the IDLE editor.

fooCount.py is a simple program that determines and outputs the amount of a value in a list. In this example, our function, fooCount, will count the number of occurrences of the word ‘foo’ in our list A, and output the value accordingly. We can test this ourselves by going to the taskbar and clicking [Run], and then [Run Module], or by simply hitting the F5 key.

The program outputs the amount of times ‘foo’ appears in our list A, and as you can see, ‘foo’ does appear in our list exactly three times. This is simple enough, but what if our list contained over one hundred elements? We would only be able to see the final output and go by that alone, without knowing the value of our count variable while the program is running in real time. This is where the debugger feature can be useful.

In the code editor window, right click on line 9, the print statement, and select [Set Breakpoint]. Breakpoints are used to establish a point at which we would like our program to pause during its execution. We can set a breakpoint on any line, and can even set multiple breakpoints if we so choose. By setting out breakpoint here, we are effectively saying that we want our program to run until it reaches this print statement, at which point the program will pause and wait for further commands.

In the IDLE main window, click on [Debug] from the taskbar, and select [Debugger]. This will bring up the Debug Control window where you can view your program running step by step. Rerun the program and keep both the Debug Control window and the main window active.

Notice how the Debug Control window now displays a line pertaining to the first line in the program. The program has been paused, and as long as we are using the Debugger, the program will always pause on the very first line of our script as shown by the Debugger. At this point, we could hit the [Go] button, which would run the rest of the program until our breakpoint is reached. For now, we will cycle through the program step by step to see what is being done on each line. Click the [Step] button to proceed to the next line of code.

The Debugger has reached line 3, the function definition of fooCount(). The program is still paused, and will remain this way until we hit the [Go] button. For now, click [Step] again to proceed to the next line.

The Debugger has skipped down to line 12 where the call to fooCount() is made. This properly illustrates the sequence of steps in which the program is run. The Debugger cannot go to the lines within the fooCount() function unless fooCount() is called, which occurs on line 12. Now that fooCount() has been called, the next step should proceed to the actual contents of the fooCount() function.

Line 4 is reached, the first line within the fooCount() function. It says to assign the value 0 to the variable count.

When you move to the next step, you will see that a watch has been added to the count variable, as shown in the bottom left corner of the window. Keep an eye on this value as the debugger moves forward in the program, as our function will be modifying this value just as we first demonstrated. As it stands, the debugger has moved to line 5, which contains the definition for our for loop that will iterate through each element in the list A.

The i variable has also been tracked, being a counter for the loop in the function. Now comes the comparison, where the value i is compared to the value ‘foo’ in order to find a match.

Since the first value in the list is not ‘foo’, the condition is not met and so we move on to the next iteration. Notice that count remains 0 since we have not found a ‘foo’ yet, and i remains 0 because it has been assigned to the first value in our list.

The next element in our list is checked, assigned to i, which happens to be 1. An occurrence of ‘foo’ has not been found yet, and so our count variable remains 0.

The same goes for the next iteration, setting i equal to 2, the third element in our list.

On the next iteration, ‘foo’ is the fourth element in the list, and so our i should become equal to ‘foo’, and our count variable should increase by 1. Since i is set to ‘foo’ our condition returns true, and so the count variable in incremented, as shown in line 7.

Notice how count is now increased by 1, as defined by our code. We could continue going through each step to show how many iterations take place in the loop, but we could also have our program run until our breakpoint. Click the [Go] button to resume normal execution of the program.

The Debugger has stopped at our breakpoint on line 9, and from our watch we can see that the value of count is 3, which is correct since there are 3 occurrences of ‘foo’ in our list A. You must click [Step] a few times after this, since the debugger will display the lower level functions that are called as a result of the print statement. These are functions that are inherently called by using the print function.

Notice on the main window that our results have been displayed to the screen.

Finally, we reach line 10, the return statement that signifies the end of our fooCount() function. Click [Step] to end the debugger.

To close the debugger, click the close button in the top right corner of the window.

This concludes the Introductory Python Debugging Tutorial.