This video covers the basics of the While Loop and its uses in Python. Like For Loops, While Loops can be used to execute a block of instructions multiple times. Unlike For Loops however, instead of looping for a certain number of iterations, While Loops will continue to execute a block of instructions as long as the expression being checked remains true.

A While Loop can be constructed using the While keyword, followed by any expression that can evaluate to true or false (or 1 or 0 alternatively). Any instructions meant to be repeated can then be placed inside the While Loop, and these instructions will continue to be run on each iteration so long as the evaluated expression remains true.

A simple example can be written as follows:

X = int(input(“Enter -1 to stop: ”))

While x != -1:

X = int(input(“You entered”,x,”. Enter -1 to stop: ”))

When we run this example, the first thing that gets executed is the input statement prompting the user to enter a value. Let’s say we enter a value that is not -1. On the following line, our While loop checks the expression x != -1. If this is true then execution should enter the block and run the line of code within the While Loop. Since our x does not equal -1 in this case, the condition evaluates to true, and thus the second input line is run. At this point we can now enter a new value for x. Let’s say we continue to keep x as a value that is not -1. After getting a new value for x, our While Loop has reached the end, and so the first iteration has been completed. Now the second iteration starts back at the top where our expression is to be evaluated again. Does x equal -1? Well no, since we entered a different value, and so the expression evaluates to true yet again. Execution will now follow through to the next line (the input line) again, and the cycle will continue with this pattern until we eventually enter a value that is -1. Let’s enter a non -1 answer again to enter the third iteration. Now the third iteration has begun, and it started by checking our current value for x just like it did before with the given expression. We can now enter another value, but this time, let’s enter -1. Once we enter -1, the next time our condition is checked, it will evaluate to false. If our condition evaluates to false, execution will exit the While Loop and continue to run the rest of our program. Since our script is short, in this case, the moment we enter x equals -1, our program will end.

It’s important to note that While Loops will always run instructions in sequential order, much like For Loops. We can see this for ourselves if we make a small modification to our code:

X = int(input(“Enter -1 to stop: ”))

While x != -1:

X = int(input(“You entered”,x,”. Enter -1 to stop: ”))

Print(“Hello”)

The statement “Hello” is always printed after we enter a new value for x. Note that “Hello” is not printed after we enter our first value for x, and this is because it is only printed after the input statement in our While Loop is executed.

Let’s do another example, this time to get multiple numeric inputs from the user, and determining the sum and average of the inputs.

Total = 0

Average = 0

Count = 0

X = int(input(“Enter a positive number. Enter -1 to stop: ”))

While x != -1:

Total += x

Count += 1

X = int(input(“Current sum:”,total,”. Enter a positive number. Enter -1 to stop: ”))

Average = total/count

Print(“Total sum:”,total)

Print(“Average:”,average)

First, we define three variables, Total for storing our sum over time, Average for storing the average, and Count for tracking the number of values that are inputted. Next, we get our first value from the user at line 4, which is stored as an integer in variable x. Our While loop begins on line 5, where the condition x != -1 is evaluated. So as long as we maintain a value for x that is not -1, every line in our While block should be run. Assuming we enter a non -1 value, our three lines in our While loop will be run in sequence. First, our current x value will be added to the total. Then our Count variable will be incremented by 1. Finally, we obtain a new value for x, and then the condition for the While Loop is checked again using the new x value we enter. This cycle will continue until we eventually enter a -1 value for x. After that, the While loop will exit, and the remaining lines in our code will run in sequence, calculating the average and then printing the results.

Let’s test our program with four inputs. Upon startup, we input the value 60 for the variable x, and this value is compared to -1 in our condition. Since 60 is not equal to -1, our condition evaluates to true, and so the first iteration of our While loop can begin. 60 is added to the total, and our Count variable is incremented by 1. New value for x is then prompted to us. Let’s input 75, and like before our condition is checked, comparing this value to -1 and returning true like before. Thus, the second iteration takes place, 70 is added to our current total, and our Count increments by 1 again. We can repeat this step for a third iteration by setting x to 80, and like before 80 is added to our total and our Count variable is incremented by 1. At this point, let’s try exiting the While loop by inputting -1 as a value for x. Our condition is checked, and since -1 is equal to -1, our expression evaluates to false. Thus, on our fourth input, the While loop will not enter a fourth iteration, and will instead exit by skipping over the block and running the next line following the end of the block. In this case, the average is calculated, and then out total sum and average are displayed. Since the last three liens of instructions are not included in the block for our While loop, they will only be run once as expected.

A very important property about While loops is that if the expression being evaluated never evaluates to false, the While loop will continue to execute statements indefinitely without ever stopping. This is known as an infinite loop, and should be avoided at all costs. To stop an infinite loop in IDLE, you can hit Control+C for force quit execution.

While loops can be nested within one another, and they can also be used in conjunction with the break command in order to exit a While loop prematurely. The mechanics behind how these work is exactly the same as with For loops. You can learn more information on nested loops and the break command by following the Python For loops video included in this tutorial series.

To summarize, While loops are used to run a sequence of instructions multiple times. They rely on a Boolean expression that either evaluates to true (1) or false (0). If the condition is true, then execution will run all instructions in the block of the loop sequentially, and then evaluate the condition again at the start of each of the subsequent iterations. If at any point during execution the expression evaluates to false, then upon entering the next iteration, execution will break from the While Loop and will continue like normal. A condition that never evaluates to false results in an infinite loop, which can cause a program to crash or freeze the processor. In the IDLE interface, an infinite loop can be forcibly stopped by hitting the control+c key in the middle of execution.