Unit 3: Conditions, If and While | **Skill Builder 2: Flag Waving**

In this lesson, you will learn about *counters* and *accumulators* by calculating the number and the total of numbers entered and determine the average of the numbers.

Objectives:

- Counter statement
- Accumulator statement
- **While** loop with a flag
- Calculating an average (mean)

You will write a program that will count and add up a set of numbers, but you do not know in advance how many numbers there will be. This lesson introduces the concept of a counter statement, an accumulator statement, a "flag" value to end a loop, and the calculation of the average of the entered numbers.

1. Begin a new Python file and name it **COUNTER**.

   Make two variables and set them both equal to zero:

   count = 0

   total = 0

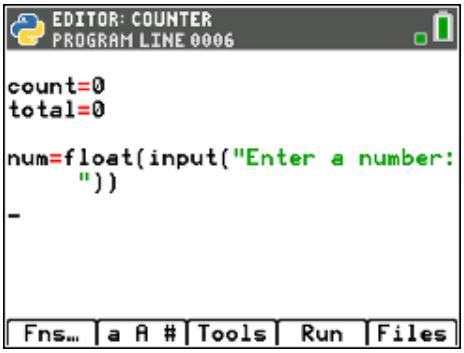
   count keeps track of how many numbers are entered and

   total keeps a running total of the numbers that have been entered.

   A third variable, **num**, stores each number entered one at a time.

   Rather than setting it equal to 0, use an **input** statement to get the first number from the user:

   num=float(input("Enter a number: "))



Remember that most of the Python programming tools are found under **<Fns…>**.

Teacher Tip: Python shortcuts:

c=c+1 can be written c+=1 (but not c++)

t=t+n can be written t+=n

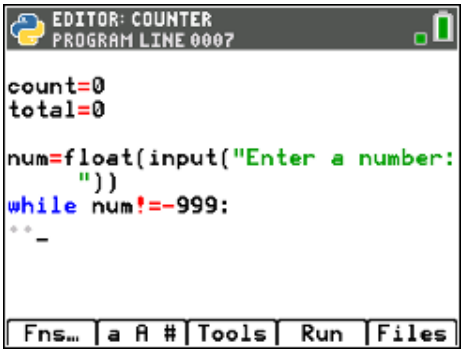
count=total=0 is allowed as well as

count, total = 0,0

The **input** statement before the **while** loop gets the loop started properly; it *initializes* the loop condition.

2. Now start a **while** loop that will end when a certain number is entered. Some options here are 0, -999 or -1. Keep in mind that this unique number cannot be one of the numbers you are trying to process. We will use -999 as our "flag" value to signal that we have finished entering numbers.

   while num != -999:



   Recall that **while** *is found on* **<Fns…> Ctl** *and* **!=** *is on all the places you can find* **==**. *Simplest is the* **[test]** menu.

Teacher Tip: Flags (or sentinels), counters and accumulators are common programming terms.

Both the “subtraction” key and the “negation” key on the keypad produce the same character since Python uses context to determine semantics.

3. The **while** loop block will:
   a) Count the number of numbers entered.
   b) Add them up.
   c) Then, ask for another number.

   Counting is performed with the statement

   count = count + 1

   which adds 1 to the variable **count** each time it is processed.

   Adding them up, or totaling, is accomplished with the statement:

   total = total + num

   “Ask for another number” requires an **input** statement like the first one:

   num=float(input(“Enter a number (-999 to end): ”))

   but here we add a message telling the user how to end the loop.

```
EDITOR: COUNTER
PROGRAM LINE 0010
count=0
total=0

num=float(input("Enter a number:
    "))
while num!=-999:
  count=count+1
  total=total+num
  num=float(input("Enter a numbe
    r(-999 to end):"))

Fns… | a A # | Tools | Run | Files
```

Tip: Copy the first **input** *statement and paste it here, then edit it.*

Be sure all three of these statements are properly indented.

The **input** *statement comes* ***last*** *in the loop body to control the loop.*

4. Test your program so far since the **while** loop is complete. Enter any number and enter -999 to end the program.

   Question: Is -999 included in the **total**?

   Answer: No!

Tip: If your program does not end when you enter -999, you can press **[on]** *to break it, go into the Editor and check your code.*

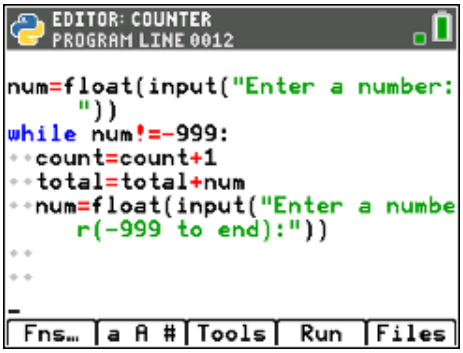
```
PYTHON SHELL
>>> # Shell Reinitialized
>>> # Running COUNTER
>>> from COUNTER import *
Enter a number:50
Enter a number(-999 to end):98
Enter a number(-999 to end):-999
>>>

Fns… | a A # | Tools | Editor | Files
```

5. We have finished with the loop. Skip a line or two and erase (backspace) the indent spaces. *Note the cursor position in this image.*

   Now is the time to process the inputted numbers. We kept track of the **count** and the **total** so now (after the loop ends) we can calculate the average.

   Try it yourself before looking at the next step.



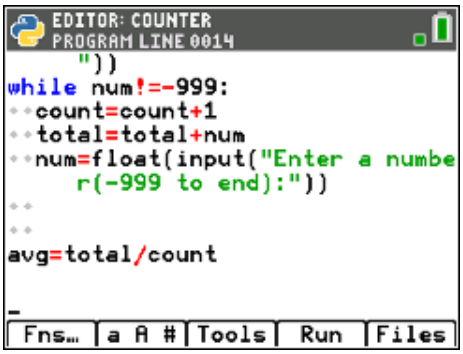
6. The statement

   avg = total / count

   calculates the average and stores the result in the variable avg.

   Ponder this: Why do we use / here and not // as in the last lesson?

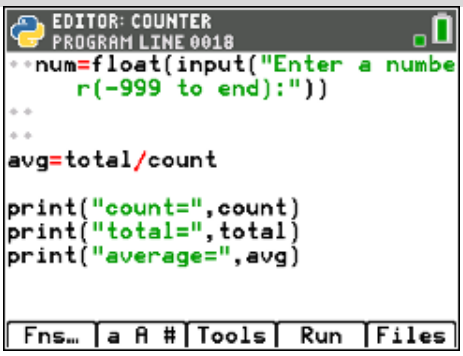
   Finally, write the **print** statement(s) that report the count, total and average to the user. You may wish to use more than one print statement. Our code is shown in the next step.



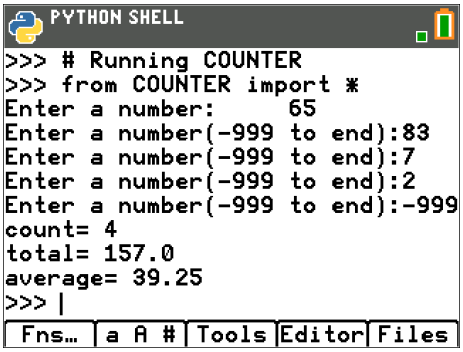
Teacher Tip: *Why do we use / here and not // as in the last lesson?*
/ is floating point division and results in decimal answers.
// is integer division and the results are integers only.

7. We used three print statements at the end of the program to report the results:

```
print("count   = ", count)
print("total   = ", total)
print("average = ", avg)
```

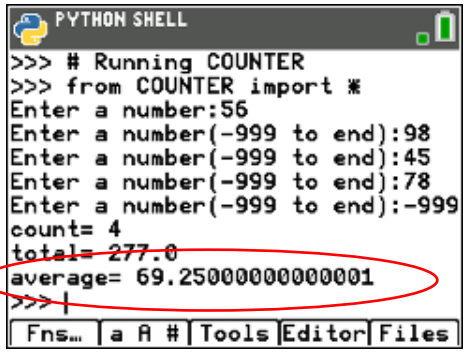


A sample run of the completed program is shown.

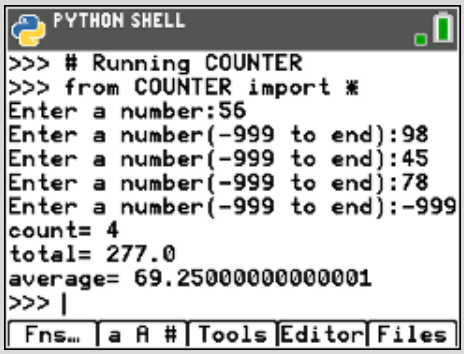


Special Note-

Sometimes you might notice the average has a lot of trailing zeros followed by a 1. Python's method of storing and computing with binary floating point numbers then converting them to decimal approximations can lead to stray decimals in some calculations. This anomaly is common with Python on virtually any computing platform and has nothing to do with the TI-84 Plus CE Python graphing calculator.



Teacher Tip: Python's method of storing and computing with floating point numbers leads to stray decimals in some calculations. This is a quirk of Python and has nothing to do with TI-84 Plus CE Python.



Challenge: As part of the input routines, display the number of numbers entered so far.