14.3. The Listener Loop

At the end of the previous section, we advised using a for loop whenever it will be known at the beginning of the iteration process how many times the block of code needs to be executed. Usually, in python, you will use a for loop rather than a while loop. When is it *not* known at the beginning of the iteration how many times the code block needs to be executed? The answer is, when it depends on something that happens during the execution.

One very common pattern is called a **listener loop**. Inside the while loop there is a function call to get user input. The loop repeats indefinitely, until a particular input is received.

Save & Run Original - 1 of 1 Show in CodeLens

```
theSum = 0
x = -1
while (x != 0):
    x = int(input("next number to add up (enter 0 if no more numbers): "))
    theSum = theSum + x

print(theSum)

```

```
10
```

Activity: 1 -- ActiveCode (ac14_3_1)

This is just our old friend, the accumulation pattern, adding each additional output to the sum-so-far, which is stored in a variable called theSum and reassigned to that variable on each iteration. Notice that theSum is initialized to 0. Also notice that we had to initialize x, our variable that stores each input that the user types, before the while loop. This is typical with while loops, and makes them a little tricky to read and write. We had to initialize it because the condition `x != 0` is checked at the very beginning, before the code block is ever executed. In this case, we picked an initial value that we knew would make the condition true, to ensure that the while loop's code block would execute at least once.

If you're at all unsure about how that code works, try adding print statements inside the while loop that print out the values of x and theSum.

14.3.1. Other uses of `while`

14.3.1.1. Sentinel Values

Indefinite loops are much more common in the real world than definite loops.

- If you are selling tickets to an event, you don't know in advance how many tickets you will sell. You keep selling tickets as long as people come to the door and there's room in the hall.
- When the baggage crew unloads a plane, they don't know in advance how many suitcases there are. They just keep unloading while there are bags left in the cargo hold. (Why *your* suitcase is always the last one is an entirely different problem.)
- When you go through the checkout line at the grocery, the clerks don't know in advance how many items there are. They just keep ringing up items as long as there are more on the conveyor belt.

Let's implement the last of these in Python, by asking the user for prices and keeping a running total and count of items. When the last item is entered, the program gives the grand total, number of items, and average price. We'll need these variables:

- `total` - this will start at zero
- `count` - the number of items, which also starts at zero
- `moreItems` - a boolean that tells us whether more items are waiting; this starts as True

The pseudocode (code written half in English, half in Python) for the body of the loop looks something like this:

```
while moreItems
    ask for price
    add price to total
    add one to count
```

This pseudocode has no option to set `moreItems` to `False`, so it would run forever. In a grocery store, there's a little plastic bar that you put after your last item to separate your groceries from those of the person behind you; that's how the clerk knows you have no more items. We don't have a "little plastic bar" data type in Python, so we'll do the next best thing: we will use a `price` of zero to mean "this is my last item." In this program, zero is a **sentinel value**, a value used to signal the end of the loop. Here's the code:

Save & Run Original - 1 of 1 Show in CodeLens

```
def checkout():
    total = 0
    count = 0
    moreItems = True
    while moreItems:
        price = float(input('Enter price of item (0 when done): '))
        if price != 0:
            count = count + 1
            total = total + price
            print('Subtotal: $', total)
```

```
        else:
            moreItems = False
    average = total / count
    print('Total items:', count)
    print('Total $', total)
```

```
Subtotal: $ 8.0
Subtotal: $ 16.0
Total items: 2
Total $ 16.0
Average price per item: $ 8.0
```

Activity: 2 -- ActiveCode (ac14_3_2)

There are still a few problems with this program.

- If you enter a negative number, it will be added to the total and count. Modify the code so that negative numbers give an error message instead (but don't end the loop) Hint: `elif` is your friend.
- If you enter zero the first time you are asked for a price, the loop will end, and the program will try to divide by zero. Use an `if` / `else` statement outside the loop to avoid the division by zero and tell the user that you can't compute an average without data.
- This program doesn't display the amounts to two decimal places. You'll be introduced to that in another chapter.

14.3.1.2. Validating Input

You can also use a `while` loop when you want to **validate** input; when you want to make sure the user has entered valid input for a prompt. Let's say you want a function that asks a yes-or-no question. In this case, you want to make sure that the person using your program enters either a Y for yes or N for no (in either upper or lower case). Here is a program that uses a `while` loop to keep asking until it receives a valid answer. As a preview of coming attractions, it uses the `upper()` method which is described in String Methods to convert a string to upper case. When you run the following code, try typing something other than Y or N to see how the code reacts:

Save & Run | Original - 1 of 1 | Show in CodeLens

```
def get_yes_or_no(message):
    valid_input = False
    while not valid_input:
        answer = input(message)
        answer = answer.upper() # convert to upper case
        if answer == 'Y' or answer == 'N':
            valid_input = True
        else:
            print('Please enter Y for yes or N for no.')
    return answer

response = get_yes_or_no('Do you like lima beans? Y)es or N)o: ')
if response == 'Y':
    print('Great! They are very healthy.')
else:
```

```
Great! They are very healthy.
```

Activity: 3 -- ActiveCode (ac14_3_3)

You have attempted 4 of 3 activities on this page

14.2. The while Statement">
14.2. The while Statement">
14.4. Randomly Walking Turtles">
✔ Completed. Well Done!
14.4. Randomly Walking Turtles">Next Section - 14.4. Randomly Walking Turtles

| Back to top