Lecture 9 — While Loops

Overview

- Loops are used to access and modify information stored in lists and are used to repeat computations many times.
- We construct loops using logical conditions: `while` loops
- We will investigate single and multiple loops

Reading: Our coverage of loops is in a different order than that of *Practical Programming*. A direct reference for reading material is Section 9.6.

Part 1: The Basics

- Loops allow us to repeat a block of code multiple times. This is the basis for many sophisticated programming tasks
- We will see two ways to write loops: using `while` loops and `for` loops
- In Python, `while` loops are more general because you can always write a `for` loop using a `while` loop.
- We will start with `while` loops first and then see how we can simplify common tasks with `for` loops later.

Basics of While

- Our first `while` loop just counts numbers from 1 to 9, and prints them.

```
i=1
while i<10:
    print(i)
    i += 1   ## if you forget this, your program will never end
```

- General form of a `while` loop:

```
block a
while condition:
    block b
block c
```

- Steps

  1. Evaluate any code before `while` (**block a**)
  2. Evaluate the `while` loop `condition`:
     1. If it is `True`, evaluate **block b**, and then repeat the evaluation of the condition.
     2. If it is `False`, end the loop, and continue with the code after the loop (**block c**).

  In other words, the cycle of evaluating the condition followed by evaluating the block of code continues until the condition evaluates to `False`.
- An important issue that is sometimes easy to answer and sometimes very hard to answer is to know that your loop will always terminate.

Using Loops with Lists

- Often, we use loops to repeat a specific operation on every element of a list.
- We must be careful to create a number that will serve as the index of elements of a list. Valid values are: 0 up to (not including) the length of list.

```
co2_levels = [ (2001, 320.03), (2003, 322.16), (2004, 328.07),\
               (2006, 323.91), (2008, 341.47), (2009, 348.92),\
               (2010, 357.29), (2011, 363.77), (2012, 361.51),\
               (2013, 382.47) ]

i=0
while i< len(co2_levels):
    print( "Year", co2_levels[i][0], \
           "CO2 levels:", co2_levels[i][1])
    i += 1
```

- Let's make some errors to see what happens to the loop.

Part 1 Practice

1. Write a while loop to count down from 10 to 1, printing the values of the loop counting variable `i` (it could be some other variable name as well).
2. Modify your loop to print the values in the list `co2_levels` in reverse order.

Accumulation of values

- Often, we use loops to accumulate some type of information such as adding all the values in a list.
- Let's change the loop to add numbers from 1 to 9.

```
i=1
total = 0
while i<10:
    total += i
    i += 1
print(total)
```

(Of course you can and should do this with the `sum()` function, but guess what that actually does!)

- Let's use a loop to add the numbers in a list.
  - Now, we will use the numbers the loop generates to index a list.
  - So, the loop must generate numbers from 0 to length of the list.

```
co2_levels = [ (2001, 320.03), (2003, 322.16), (2004, 328.07),\
               (2006, 323.91), (2008, 341.47), (2009, 348.92),\
               (2010, 357.29), (2011, 363.77), (2012, 361.51),\
               (2013, 382.47) ]

i=0
total=0
while i< len(co2_levels):
    total += co2_levels[i][1]
    i += 1

print("Total co2_levels is", total)
```

- Let's have a more interesting example. Be very careful not to use an incorrect index for the list.
  - Count the number of CO2 values in the list that are greater than 350.
  - Calculate and print the percentage change in each measurement year from the previous measurement year.
  - Determine the years in which the CO2 levels dropped compared to the previous measurement. Output just these years.

Part 2 Practice

1. Suppose we wanted to print the first, third, fifth, etc. elements in a list. Write code to accomplish this.

```
months=['jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec']
```

2. Now, use a similar loop code to print a little evergreen tree.

```
    *
   ***
  *****
 *******
*********
   ***
   ***
```

3. Try this later: change your loop to work for any size evergreen.

Loops that end on other conditions

- Here is a while loop to add the non-zero numbers that the user types in.

```
total = 0
end_found = False

while not end_found:
    x = int( input("Enter an integer to add (0 to end) ==> "))
    if x == 0:
        end_found = True
    else:
        total += x

print(total)
```

- We will work through this loop by hand in class.

Multiple nested loops

- Loops and if statements can both be nested.
- We've already seen this for if statements.
- Here's an example where we print every pair of values in a list.

- First solution:

```
L = [2, 21, 12, 8, 5, 31]
i = 0
while i < len(L):

    j = 0
    while j < len(L):
        print(L[i], L[j])
        j += 1

    i += 1
```

- This solution prints the values from the same pair of indices twice - e.g.the pair 21, 12 is printed once when *i=1*, *j=2* and once when *i=2*, *j=1*.
- How can we modify it so that we only print each pair once?
- What has to change if we don't want to print when *i==j*?
- Finally, we will modify the resulting loop to find the two closest values in the list.

Summary

- Loops are especially useful for iterating over a list, accessing its contents, and adding or counting the values from a list. This is done in the `sum()` and `len()` functions of Python.
- Each loop has a stopping condition — the boolean expression in the *while* statement. The loop will end when the condition evaluates to *True*.
- If the stopping condition is never reached, the loop will become "infinite".
- Often a counter variable is used. It (a) is given an initial value before the loop starts, (b) is incremented (or decremented) once in each loop iteration, and (c) is used to stop the loop when it reaches the index of the end (or beginning) of the list.
- We will demonstrate a simple way to write these common loops with a `for` loop in the next lecture.