While Loops

CMSC 104 Section 02
October 21, 2025

Administrative Notes

We’ll go over Project 2 today

- It will be due Monday, November 3

Classwork 6 will be in class on Thursday

Loops

Loops are used to implement “iterative” control flow

- Where you do a set of things multiple times
    - Either a set number of times
    - Or until something is True or False

Two types of loops so far…

So far in class we've learned two types of loops:

- "For each" loops
  - Simple syntax and structure
  - Only useful when you want to go through a list
  - And do something with each element in the list, exactly once
- "For i" loops
  - More general; slightly more complicated syntax
    - `for i in range(begin, stop, step):`
  - Great when you want to do something a set number of times

And now the most general type of loop

“While” loops

- General syntax:
  - `while (boolean-expression):`
    - `# code indented underneath the while statement`
- Semantics:
  - At the beginning of the loop, evaluate the boolean-expression
    - If it’s True, execute the code in the body of the loop
    - If it’s False, skip the body of the loop and move on to the next line of code AFTER the loop - the next line not indented under “while”

Why is this more general?

You’re not limited to executing a loop once for each item in a list; or a set number of times

You may not know how many times your loop will be executed

- It runs until the user types “quit”
- It runs until there is no more data left in a file
- It runs until something else happens

It may run zero times

- This is quite common for loops that check user input

Loops that run zero times

Suppose you ask the user to enter a positive integer

- You check to make sure that the user entered a positive integer
- While the user enters things other than a positive integer, you throw an error message and ask the user to try again
- If the user enters a positive integer the first time, your loop will never execute
    - This is desired behavior

Sentinel loops

A “sentinel” is a value that you are looking for, and take some action upon seeing

- E.g., write a loop that will run until the user enters some value, and then stops
- Here, “Q” is the “sentinel” value
-

Sentinels when reading from a file

If you’re reading data from a file, the end of the file is marked with an empty line - “”

- Since all other lines contain at least a newline character - “\n” - this is a unique way to tell the programmer that the file has no more data
- So you can use a loop with readline() and read one line at a time. Stop when you read an empty line

Infinite loops

An “infinite loop” is a loop that will literally run forever, unless you take some action outside of the program to stop it.

There are very rare instances when this is what you want - but normally, this is considered a “very bad thing”

It is very, very difficult to write an infinite loop using “for”

- You have to want to do it

But it is easy to write an infinite loop using “while”

- You can forget to increment a variable you’re using in the boolean expression
- You can design your boolean expression to be “not quite correct”
- Etc.

Examples of infinite loops

When your program has an infinite loop

How to tell that your program has an infinite loop

- It keeps printing output long after it should have stopped
    - if there’s a print() statement in the loop
- It looks like nothing is happening for a long time
    - If there’s no print() statement

What to do about it

- Hold down the Control key and hit c (Ctrl-c)
    - Either uppercase or lowercase C should work; lowercase is easier
- If you’re on a Mac and that doesn’t work, try Command-c
- If that doesn’t work, you’ll have to force IDLE to quit in the Operating System
    - Task Manager in Windows; Force Quit with Mac

Breaking out of a loop in the middle

(Caution: this is generally considered bad programming practice. Most of the time you can write a loop so that it doesn’t break out in the middle. But sometimes it’s necessary.)

Why is this the most complex loop to write?

Because it’s easy to get into an infinite loop

- Variables in the boolean-expression are not automatically updated in the loop; you have to update them yourself
    - With both types of “for” loops, indices are updated automatically
- If you don’t write the boolean-expression correctly, it may never be False

“With great power comes great responsibility”

Why are while loops the most general type?

True statement:

- Any loop in Python can be written as a “while” loop
- Not every loop in Python can be written as a “for i” or “for each” loop
    - Loops where the number of times the loop gets executed depends on some unknown external influence