For Loops and While Loops

September 29, 2021

Administrative Notes

The sample exam is finally available on the class GitHub repo.

- We'll go over it in class on Monday, October 11

Next week's lectures will cover additional string operations that we have not yet covered.

- That will be included on the test
- We won't get to functions until AFTER exam 1

Homework 3 is due Friday night at midnight

- We'll talk about that now

Main programs and functions

We ask you to start your program with the following statement:

```
if __name__ == “__main__”:
    #indent your code under here
```

Let’s explain what this means and why you do it

Note: functions won’t come until later, but we want you to get in the habit

for Loops

“For each” loops -

- A special case, when you want to do something with each element in a list, exactly once
  - Print each element; check to see if it meets a condition like ending in “a”; multiplying each element by 6; ...
- And you don’t want to change the element values themselves
- Examples:

for state in STATES

for i loops...

A more general case of the for loop

- Uses “range” to control movement through the loop & list
- Forward, backward, partway through the list,....
- Allows you to change elements in the list while you’re working through the list

Some examples:

Now: while loops

The most general type of loop in Python

- Everything you can do with a for loop can be done with a while loop
- The opposite is *not* true

But also risky

- While loops are the easiest type to make mistakes in
- “Infinite loops” - loops that will literally never end without outside interference

- “for” loops are used when you know how many times you want to iterate through some code
  - You know the specific number of times
  - You know you want to iterate a number of times that is the value of a specific variable, such as the length of a list
- “while” loops are the only loop used when you don’t know how many times you will iterate through code

Topics

While loops in general

Sentinel loops

Boolean flags

Syntax of a while loop

```
while boolean-condition-is-true:

    Code to be executed
```

Remember indentation. All the code that is indented underneath the “while” statement is executed as part of the loop. When you unindent, that code is no longer part of the loop.

Set the value of your boolean condition. Unlike with “for” loops, Python does not automatically set a value for a new variable used in the boolean condition of a “while.”

Examples:

```
age = 0;
while (age < 18):
    age = int(input(“enter your age in years: “))
    print (“If you’re 18 or older you should vote”)
print (“that’s the end of our story”)
```

What happens if we leave out the initial age = 0 statement? The loop fails because age has no value.

How many times will this loop be executed? We don’t know - until the user cooperates

‘Priming read’

```
age = int(input(“enter your age in years: “))
while (age < 18):
    age = int(input(“enter your age in years: “)}
    print (“If you’re 18 or older you should vote”)
print (“that’s the end of our story”)
```

“Priming” - refers to getting an initial value before executing the loop

- If the user enters 35 - the loop never executes

Example: factorial

```
# compute 10! Using a while loop

product = 1

factor = 1

while factor <= 10:

    product *= factor    #or product = product * factor

    factor += 1
```

Common programming errors 1: Loop is never executed

Suppose we want to print out all of the even numbers between 2 and 100 inclusive. Why won't this loop work?

```
num = 1

while num % 2 == 0:

    print(num)

    num += 2
```

It is perfectly acceptable to write a loop that may never be executed, due to other conditions in your code.

But be sure that that's really what you want

A loop that never executes:

```
age = int(input("please enter your age in years: "))
while (age < 0) and (age > 100):
    print("Age must be between 0 and 100 inclusive ")
    age = int(input("please enter your age in years: "))
```

If the user enters, say, 21 at the first prompt, the boolean condition is false at the start, and the code under the while is never executed.

That's perfectly fine. Just make sure that it's really what you wanted.

Common programming errors II: infinite loops

An infinite loop is one that never stops executing, because the boolean condition never becomes false.

Common causes:

- You never change a variable used in the boolean condition
- You plan to stop when a variable takes on a value that it will never take on

The code will never stop on its own. The program is only stopped by external action - you shut off the program or run out of resources. On gl.umbc.edu and similar machines, you can hit “Control” and “c”.

Note: infinite loops can occur with “for” loops, but you have to really work hard to make that happen. They’re rare.

Infinite loop examples

```
grade = ""
name  = ""
while name != "Hrabowski":
    # get the user's grade
    grade = input("What is your grade? ")
print("You passed!")
```

```
cookiesLeft = 50

while cookiesLeft > 0:
    # eat a cookie
    cookiesLeft = cookiesLeft + 1
```

```
#print all the positive odd numbers
#less than 100

num = 1
while num != 100:
    print(num)
    num = num + 2
```

Sentinel Loops

A loop that runs until a specific value is encountered

A ‘sentinel’ is a value that denotes the end of a data set.

Simple example: letting the user input data. The user inputs “Q” to indicate that it is time to quit. “Q” is the sentinel value

Code - see next slide

Sentinel loop - example

```
birthdate = input("Enter your month and day of birth as mm/dd. Enter 'Q' to
quit.")
while birthdate != "Q":
    month_and_day = birthdate.split('/')
    for i in range(2):
        month_and_day[i] = int(month_and_day[i])
    day_of_year = 30 * month_and_day[0] + month_and_day[1]
    print(f"Your birthday is the {day_of_year:5d} day of the year")
    birthdate = input("Enter your month and day of birth as mm/dd. Enter 'Q'
    to quit.")


        :
```

Boolean flags

You have to have a boolean condition in your while loop, so what's a "Boolean flag?"

- It's a boolean variable that you explicitly set to true or false and use that to end a while loop

Boolean Flag

```
prompt = "Tell me something cool: "
prompt += "\nEnter 'quit' to end the program"
active = True
while active:
    message = input(prompt)
    if message == 'quit':
       active = False
    else:
       print(message)
```

NOT a Boolean Flag

```
prompt = "Tell me something cool: "
prompt += "\nEnter 'quit' to end the program"
message = input(prompt)
while message != "quit":
    print(message)
    message = input(prompt)
```