Computer Science 1 — CSci 1100
Lecture 11 — While Loops

Overview

- While loops
- Break and continue
- Examples:
  - Counting and summations; simple input
  - Random walk
  - Amino acid search
- Infinite loop
- Input loops
- Controlling loops through `break`, `continue` and `enumerate`
- Difference between `for` and `while`.

Reading: *Practical Programming*, rest of Chapter 7.

Part 1: The Basics

- `for` loops tend to have a fixed number of iterations computed at the start of the loop
- `while` loops tend to have an indefinite termination, determined by the conditions of the data
- Most Python `for` loops are easily rewritten as `while` loops, but not vice-versa.
  - In other programming languages, `for` and `while` are almost interchangeable, at least in principle.

Basics of While

- Our first `while` loop just adds digits and could easily be a `for` loop

```
i=1
sum = 0
while i<10:
    sum += i
    i += 1
print sum
```

- General form of a `while` loop:

```
while condition:
    block
```

- Steps

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

  In other words, the cycle of evaluating the condition followed by evaluating the block of code continues until the condition evaluates to `False`.

While Loop to Add Numbers From Input

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

```
sum = 0
end_found = False

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

print sum
```

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

Part 1: Exercises

1. Write a while loop to output the numbers from 9 down to and including 0.
2. Write a function that returns the index of the first negative number of the list passed to it as an argument. It should use a while loop and it should return -1 if there are no negative numbers in the list.

Part 2: Controlling While Loops

- We can control while loops through use of
  - `break`
  - `continue`
- We need to be careful to avoid infinite loops

Using a Break

- We can rewrite the above example to terminate the loop immediately upon seeing the 0 using Python's `break`:

```
sum = 0
while True:
    x = int( raw_input("Enter an integer to add (0 to end) ==> "))
    if x == 0:
        break;
    sum += x

print sum
```

- `break`
  - sends the flow of control immediately to the first line of code outside the current loop, and
- The while condition of `True` essentially means that the only way to stop the loop is when the condition that triggers the `break` is met.

Continue: Skipping the Rest of a Loop

- What if `yelp.txt` contains blank lines with no data? We'd like to skip over these.
- We can do this by telling Python to `continue` when it sees a blank line:

```
file = open('yelp.txt','r')
restaurants = []
for line in file:
    p_line = p_line.strip()
    if len(p_line) == 0:
        continue
    restaurants.append( parse_line(line) )
```

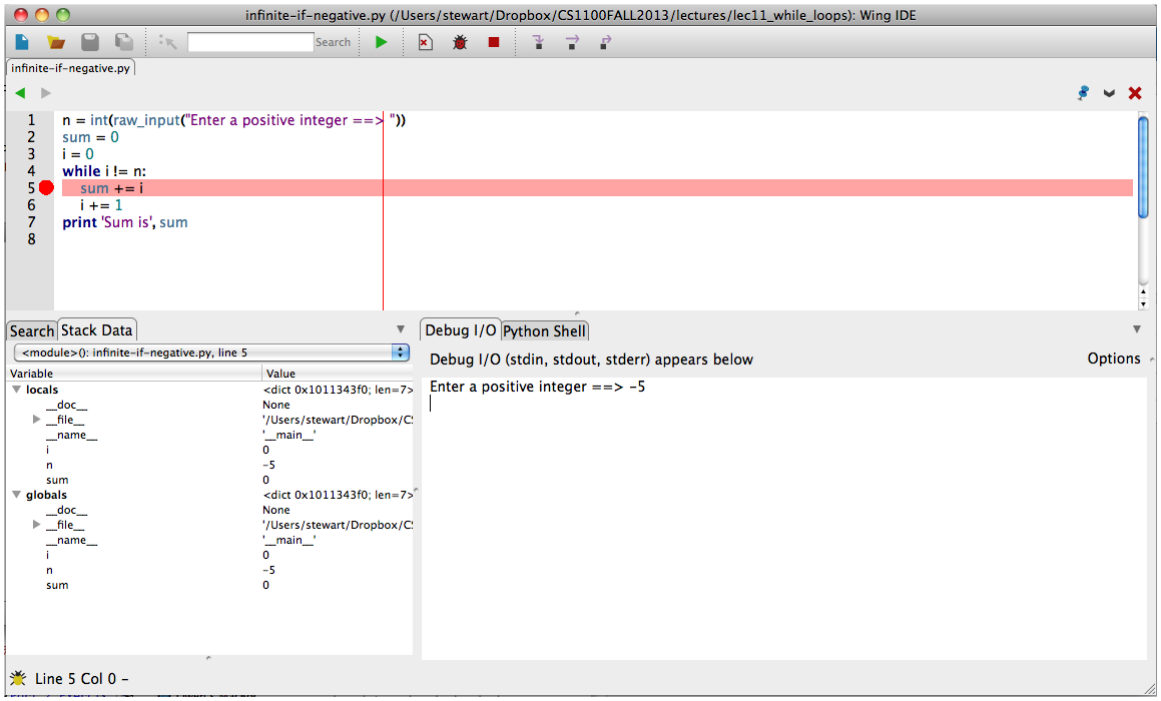
- When it sees `continue`, Python immediate goes back to the `while` condition and re-evaluates it, skipping the rest of the loop.
- Any `while` loop that uses `break` or `continue` can be rewritten without either of these.
  - Therefore, we choose to use them only if they make our code clearer.
  - A loop with more than one continue or more than one break is often unclear!
- This particular example is probably better without the `continue`.
  - Usually when we use `contnue` the rest of the loop would be much longer, with the condition that triggers the `continue` tested right at the time.

Infinite Loops and Other Errors

- One important danger with `while` loops is that they may not stop!
- For example, it is possible that the following code runs "forever". How?

```
n = int(raw_input("Enter a positive integer ==> "))
sum = 0
i = 0
while i != n:
    sum += i
    i += 1
print 'Sum is', sum
```

- How might we find such an error?
  - Careful reading of the code
  - Insert print statements
  - Use the Wing IDE debugger.
- We will practice with the Wing IDE debugger in class, using it to understand the behavior of the program. We will explain the following picture



and note the use of

- The hand, bug and stop symbols on the top of the display, and
- The Debug I/O and Stack Data at the bottom of the display.

Part 2 Exercises

1. Given two lists `L1` and `L2` measuring the daily weights (floats) of two rats write a `while` loop to find the first day that the weight of rat 1 is greater than that of rat 2.

2. Do either of the following examples cause an infinite loop?

```
import math
x = float(raw_input("Enter a positive number -> "))
while x > 1:
    x = math.sqrt(x)
print x
```

```
import math
x = float(raw_input("Enter a positive number -> "))
while x >= 1:
    x = math.sqrt(x)
print x
```

Example: Random Walk

- Many numerical simulations, including many video games involve random events.
- Python includes a module to generate numbers at random. In particular,

```
import random
```

```
# Print three numbers randomly generated between 0 and 1.
print random.random()
print random.random()
print random.random()

# Print a random integer in the range 0..5
print random.randint(0,5)
print random.randint(0,5)
print random.randint(0,5)
```

- We'd like to use this to simulate a "random walk":
  - Hypothetically, a (very drunk) person takes a step to the left or a step to the right, completely at random (equally likely to go left or right), in one time unit.
  - If the person is on a platform with N steps and the person starts in the middle, this process is repeated until s/he falls off (reaches step 0 or step $N + 1$)
  - How long does this take?
- Many variations on this problem appear in physical simulations.
- We can simulate a steps in two ways:
  1. If `random.random()` returns a value less than 0.5 step to the left; otherwise step to the right.
  2. If `random.randint(0,1)` returns 0 then step left; otherwise, step right.
- We'll write the code in class, starting from the following:

```
import random

#  Print the output
def print_platform( iteration, location, width ):
    before = location-1
    after =  width-location
    platform = '_'*before + 'X' + '_'*after
    print "%4d: %s" %(iteration,platform),
    raw_input( '  <enter>')   # wait for an <enter> before the next step

################################################################################

if __name__ == "__main__"
    # Get the width of the platform
    n = int( raw_input("Input width of the platform ==> ") )
```

Exercise: Evaluating a DNA Sequence

Consider the DNA sequence, represented by the string:

```
seq = 'ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCC' \
      'CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC' \
      'CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG' \
```

```
'AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC' \
'CTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG' \
'TTTAATTACAGACCTGAA'
```

3-letter subsequences encode amino acids. For example, ACA is Threonine, and AGA is Arginine.

1. As an **exercise**, ignoring the existence of the `find` function for strings, write a function that uses a while loop to find and return the index of the first occurrence of a particular amino acid, represented by a three-letter string, in a DNA sequence. It should return -1 if the amino acid is not there. The format of the function is

   ```
   def find_amino( amino, dna_seq ):
   ```

2. How could you solve the same problem using a `for` loop and a `break`, or with no `break`?

3. Revise your code so that `amino` can be an arbitrarily long string instead of just three characters!

Summary

- While loops should be used when the termination conditions must be determined during the loop's computation.
- Both for loops and while loops may be controlled using break and continue, but don't overuse these.
- While loops may become "infinite"
- Use a debugger to understand the behavior of your program and to find errors.
- While loops were illustrated using the following examples:
  - Input / sum loop
  - Random walk
  - Counting amino acids in a DNA sequence