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.

While

- 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.

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 (not shown here) 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 (not shown here) 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`.

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.

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.

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
  - if there are nested loops it only ends the current loop, not any outer loops.

Example: Random Walk

- Many numerical simulations including some computer/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()
```

- We'd like to use this to simulate a "random walk" — hypothetically the steps, left or right, that a very drunken person might take.
- Sometimes this is posed that the person is on a platform and if s/he takes too many steps left or right and s/he will fall off. We'd like to know how many steps.
- 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

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

Continue: Skipping the Rest of a Loop

- What if the Lab 5 input file 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('lab5businesses.txt','r')
for line in file:
    p_line = p_line.strip()
    if len(p_line) == 0:
        continue
    p_line = parse_line(line)
    #  other processing code to be added here
```

- 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 very likely to be unclear!

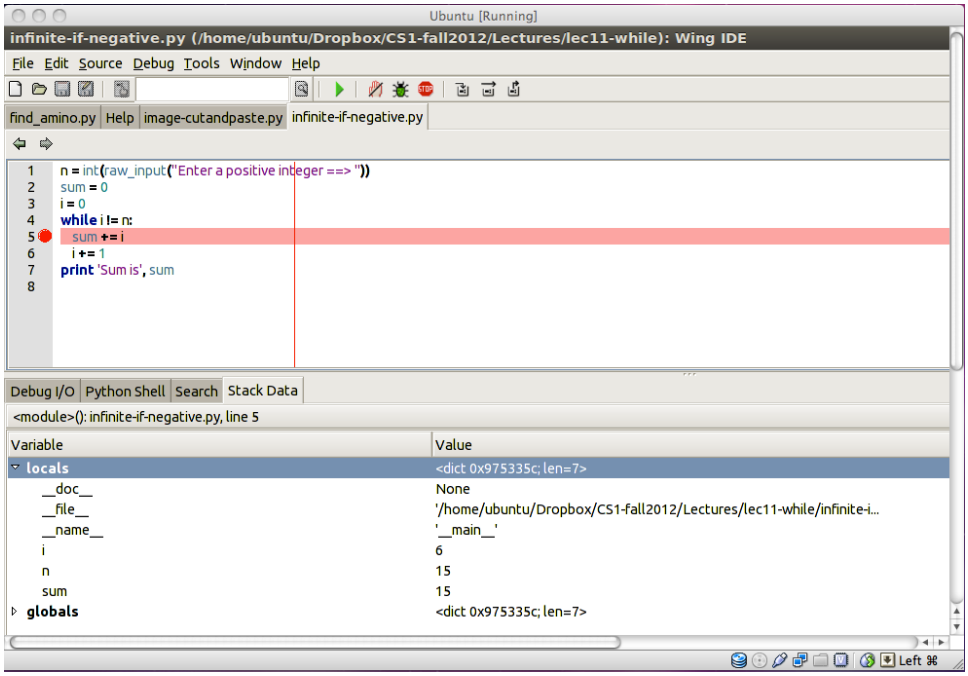
Infinite Loops and Other Errors

- 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.

More Sophisticated Example: Evaluating a DNA Sequence

- Consider the DNA sequence, represented by the string:

```
seq = 'ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCC' \
      'CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC' \
      'CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG' \
      'AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC' \
      'CTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG' \
      'TTTAATTACAGACCTGAA'
```

- 3-letter subsequences encode amino acids. For example, ACA is Threonine, and AGA is Arginine.
- Our goal is to be able to locate and count occurrences of any particular amino acid.

Exercise

1. 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. Does your solution look like other problems we have solved in this or in previous lectures?

Finding All Occurrences

- Continuing with this problem, we want to count all occurrences of an amino acid.
- We can use the function we just wrote as an exercise, but a better function is already available for us as the `find` function for `str` objects.
- After playing around with `find`, we will solve our occurrence counting problem using `find` as a tool. We'll fill in the following code:

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

  ```
  threonine = 'ACA'
  occurs = count_occurrences( seq, threonine )
  print "The threonine sequence %s occurs %d times" %(threonine, occurs)

  arginine = 'AGA'
  occurs = count_occurrences(seq,arginine)
  print "The threonine sequence %s occurs %d times" %(arginine, occurs)
  ```

- After writing this, we'll test it and use the debugger to perhaps find and fix any mistakes.

Extra Practice Problems

Before starting HW 4 and as practice problems for Test 2, work the following problems:

1. Write Python code to generate the following ranges
   (a) $(100, 99, 98, \ldots, 0)$
   (b) $(55, 53, 51, \ldots, -1)$
   (c) $(3, 5, 7, 9, \ldots, 29)$
   (d) $(-95, -90, -85, \ldots, 85, 90)$
2. Write two versions of a function called `count_in_range` that counts the number of values in a list that are between x0 and x1. The function begins with

   ```
   def count_in_range( v, x0, x1):
   ```

   One version of the function should use a `for` loop and one version should use a `while` loop.
3. Write a `for` loop that outputs the indices and the contents of any list. For example, the list

```
v = [ 'car', 15, 'bicycle', 'xray', -3.14 ]
```

should be output as

```
0: car
1: 15
2: bicycle
3: xray
4: -3.14
```

Do this both with and without an `enumerate` statement.

4. Modify the example image code from Lecture 10 to do the following:
   (a) Show an image sideways
   (b) Invert the intensities of an image
   (c) Interchange the red and green components of an image
   (d) Resize an image so that it is the same height but half the width.
5. Modify the random walk code in each of the following two ways separately:
   (a) Instead of "falling off" when the location variable reaches the minimum value, the walker can only either stay in place or move to a higher location (with equal probability). This simulates the effects of placing a wall at one side of the platform.
   (b) Add a "momentum" to the random walk so that at each step the walker has a 60% probability of continuing in the direction s/he just stepped and a 40% probability of going back in the opposite direction. The first step can go in either direction with equal probability.

   In each case experiment with this to see the effects of the changes. We could write a larger Python program to gather a wide variety of statistics on the behavior of the random walk as we change the size of the platform and, in the momentum version, as we change the probabilities.
6. Since amino acids are encoded by three consecutive values of bases A, C, G, T, only matches against the DNA sequence that start at indices 0, 3, 6, etc. should be considered true matches. Modify the occurrence counting code to only count true matches.

Summary

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