CS 1110
Lecture 19: While loops

Announcements

Prelim 2 conflicts info due on CMS by midnight tonight.
S/U students do not take prelim 2.

A4 released; note suggested milestone deadlines

Reading for next time (important, and requires concentration)

Slides by D. Gries, L. Lee, S. Marschner, W. White
Iteration: Doing things repeatedly

1. Process each item in a sequence
   - Compute aggregate statistics for a dataset, such as the mean, standard deviation, etc.
   - Send everyone in a social-media group an appointment time

2. Perform $n$ trials or get $n$ samples
   - Draw $n$ cards to make a poker hand
   - Run a protein-folding simulation for $10^6$ time steps

3. Do something an unknown number of times
   - CUAUV team, vehicle keeps moving until reached its goal
Beyond Iteration on Iterables: The while-loop

<initialization>
while <condition>:
    statement 1
    ...
    <stmt(s) that could invalidate condition>
    ...
    statement n

Four "loopy" questions for reasoning about correctness:
1. How does it start?
2. When is there still work to do? (what's the opposite of done?)
3. How do we make progress?
4. How do we make sure the loop body does the right thing?
While-Loops and Flow

1. `print 'Before while'
2. `i = 0`
3. `while i < 3:`
4. `print 'Start loop i', i`
   5. `i += 1`
6. `print 'End loop'`
7. `print 'After while'`

Output:

Before while [from line 1]
Start loop i 0 [from 4]
End loop [from 6]
Start loop i 1 [from 4]
End loop [from 6]
Start loop i 2 [from 4]
End loop [from 6]
After while [from 7]
Beyond Iteration on Iterables: The **while-loop**

```plaintext
<initialization>
while <condition>:
    statement 1
    ...
    <stmt(s) that could invalidate condition>
    ...
    statement n
```

Four "loopy" questions for reasoning about correctness:
1. How does it start?
2. When is there still work to do? (what's the opposite of done?)
3. **How do we make progress? (the "increment")**
4. How do we make sure the loop body does the right thing?
**while Versus for**

# process range b..c
k = b

**while** k <= c:
    process k
    k = k+1

# process range b..c
for k in range(b,c+1):
    process k

For-loops handle "book-keeping" implicitly in the execution's position in the for-statement list.

While-loops aren't inherently based on lists.
print "\nHope you had a happy Valentine's Day!"

raw_input("Press 'return' to pick a flower. ")
f = Flower()

print "New flower picked. "

while f.num_petals > 0:
    raw_input('Press "return" to pluck a petal. ')
    pluck(f)
Case: easier to phrase stopping condition

# store squares up to N
seq = []
n = math.floor(math.sqrt(N)) + 1
for k in range(n):
    seq.append(k*k)

# store squares up to N
seq = []
k = 0
while k*k < N:
    seq.append(k*k)
    k = k+1

A for-loop requires that you know where to stop the loop **ahead of time**

A while loop can use complex expressions to check if the loop is done
**Case: no list-based expression of stopping**

```python
def sqrt(c):
    """Return: square root of c
Uses Newton’s method
Pre: c >= 0 (int or float)"
    x = c/2.0
    # Check for convergence
    while abs(x*x - c) > 1e-6:
        # Get x_{n+1} from x_n
        x = x / 2 + c / (2*x)
    return x
```

- Want square root of $c$
  - Make poly $f(x) = x^2 - c$
  - Want root of the poly $(x$ such that $f(x)$ is 0)

- Use Newton’s Method
  - $x_0 = \text{GUESS} (c/2??)$
  - $x_{n+1} = x_n - f(x_n)/f'(x_n)$
    - $= x_n - (x_n x_n - c)/(2x_n) = x_n - x_n/2 + c/2x_n$
  - Stop when $x_n$ good enough

You'll see a (simple) termination like this in A4.
Case: modification of sequence desired

# Remove all 3's from list t
while 3 in t:
    t.remove(3)

# Replace all 5's in list t with 4's
while 5 in t:
    i5 = t.index[5]
    t[i5] = 4
Note on Ranges

- m..n is a range containing n+1-m ints: m, m+1, ..., n
  ("followers minus first")
  - 2..5 contains 2, 3, 4, 5.  Contains 5+1 – 2 = 4 values
  - 2..4 contains 2, 3, 4.  Contains 4+1 – 2 = 3 values
  - 2..3 contains 2, 3.  Contains 3+1 – 2 = 2 values
  - 2..2 contains 2.  Contains 2+1 – 2 = 1 values
  - 2..1 contains ???

- The notation m..n always implies that m <= n+1
  - So you can assume this even if we do not say it
  - If m = n+1, the range has 0 values
Note on Ranges

• m..n is a range containing the n+1-m values m, m+1, ... n
  - 2..5 contains 2, 3, 4, 5. Contains 5+1 – 2 = 4 values
  - 2..4 contains 2, 3, 4. Contains 4+1 – 2 = 3 values
  - 2..3 contains 2, 3. Contains 3+1 – 2 = 2 values
  - 2..2 contains 2. Contains 2+1 – 2 = 1 values

What does 2..1 contain?
A: nothing
B: 2,1
C: 1
D: 2
E: something else
## Patterns for Processing Integers

### range a..b-1

<table>
<thead>
<tr>
<th>i = a # i = start of undone</th>
</tr>
</thead>
<tbody>
<tr>
<td>while i &lt; b:</td>
</tr>
<tr>
<td>process integer i</td>
</tr>
<tr>
<td>i = i + 1</td>
</tr>
</tbody>
</table>

# store in count # of '/'s in string s

count = 0

i = 0 # count: num / in s[0..i-1]

while i < len(s):
  if s[i] == '/':
    count = count + 1
  i += 1

# count is # of '/'s in s[0..s.length()-1]

### range c..d

<table>
<thead>
<tr>
<th>i = c # i = start of undone</th>
</tr>
</thead>
<tbody>
<tr>
<td>while i &lt;= d:</td>
</tr>
<tr>
<td>process integer i</td>
</tr>
<tr>
<td>i = i + 1</td>
</tr>
</tbody>
</table>

# Store in v the sum: 1.0/k for k in 1..n
#
# 1/1 + 1/2 + ... + 1/n

v = 0

i = 1 # v is sum of 1.0/k for k in 1..i-1

while i <= n:
  v = v + 1.0 / i
  i += 1

# v = 1/1 + 1/2 + ... + 1/n