Announcements

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

A4 planned release: by this morning
(if it hasn’t already happened)

Reading for next time (important, and requires concentration)

Beyond Iteration on Iterables:
The while-loop

```python
<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?

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
   - CUAV team, vehicle keeps moving until reached its goal

While-Loops and Flow

```python
print 'Before while'
count = 0
i = 0
while i < 3:
    print 'Start loop', i
    count = count + i
    i = i + 1
print 'End loop'
print 'After while'
```

Output:
- Before while
- Start loop 0
- Start loop 1
- Start loop 2
- End loop
- After while

While Versus for

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

```python
# process range b..c
k = b
while k <= c:
    process 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

while Versus for

```python
# table of squares to N
seq = []
n = math.floor(math.sqrt(N)) + 1
for k in range(n):
    seq.append(k**2)
```

```python
# table of squares to N
seq = []
k = 0
while k**2 < N:
    seq.append(k**2)
    k += 1
```

CS1110 Spring 2014: While Loops
Cases to Use while: termination without counters

def sqrt(c):
    """Return: square root of c
    Uses Newton’s method
    Pre: c >= 0 (int or float)"""
    x = c/2
    # 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

You’ll see a (simple) termination like this in A4.

Cases to use while

Patterns for Processing Integers

Note on Ranges

Note on Ranges

A: nothing
B: 2,1
C: 1
D: 2
E: something else

A pattern for processing integers:

```python
range a..b-1
```

```python
i = a
while i < b:
    process integer i
    i = i + 1
```

```python
range c..d
```

```python
i = c
while i <= d:
    process integer i
    i = i + 1
```

• 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_n = GUESS (c/2 ??)
    x_{n+1} = x_n – f(x_n)/f’(x_n)
    = x_n – x_n^2 + c/2
    = x_n/2 + c/2
  • Stop when x_n good enough

A: nothing
B: 2,1
C: 1
D: 2
E: something else

A: """"Sometimes you want to modify the sequence"

```python
# Remove all 3’s from list t
while 3 in t:
    t.remove(3)
```

A: nothing
B: 2,1
C: 1
D: 2
E: something else

A: """"Note on Ranges"

```python
# Store in v the sum:
# 1/1 + 1/2 + …+ 1/n
v = 0
i = 1
while i <= n:
    v = v + 1.0 / i
    i += 1
```

A: nothing
B: 2,1
C: 1
D: 2
E: something else