# 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 planned release: by this morning**
(if it hasn't already happened)

**Reading for next time (important, and requires concentration)**
http://www.cs.cornell.edu/Courses/cs1110/2013sp/materials/loop_invariants.pdf

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

   for x in sequence: process x
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

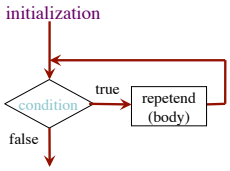
   for x in range(n): do next thing
3. Do something an unknown number of times
   - CUAUV team, vehicle keeps moving until reached its goal

   ????


Richard King

## 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

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

Output:

```
Before while
Start loop i 0
End loop
Start loop i 1
End loop
Start loop i 2
End loop
After while
```

## while Versus for

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

```
# process range b..c
k = b
while k <= c:
    process k
    k = k+1
```

## while Versus for

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

```
# table of squares 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

## 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
```

- 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**
  - $x_0$ = GUESS ($c$/2??)
  - $x_{n+1} = x_n - f(x_n)/f'(x_n)$
    $= x_n - (x_n x_n - c)/(2x_n)$
    $= x_n - x_n/2 + c/2x_n$
    $= x_n/2 + c/2x_n$
  - Stop when $x_n$ good enough

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

## Cases to use while

Sometimes you want to modify the sequence

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

## 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

## Note on Ranges

- m..n is a range containing n+1-m values
  - 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 that even if we do not say it
  - If m = n+1, the range has 0 values

## Patterns for Processing Integers

**range a..b-1**

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

**range c..d**

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

```
# store in count # of '/'s in string s
count = 0
i = 0
while i < len(s):
    if s[i] == '/':
        count= count + 1
    i += 1
# count is # of '/'s in s[0..s.length()-1]
```

```
# Store in v the sum:
# 1/1  + 1/2 + ...+ 1/n
v = 0    # call this 1/0 for today
i = 0
while i <= n:
    v = v + 1.0 / i
    i += 1
# v= 1/1  + 1/2 + ...+ 1/n
```