CS 1110

Lecture 18: While loops

Announcements

Prelim 2 conflicts

If you have a conflict you need to submit the information in CMS. We need a little more information than for Prelim 1—please see the Exams page of the CS1110 website.

Instructor travel

Over the next three weeks Profs. Lee and Marschner will be traveling on and off. Instructor office hours are unaffected, though there will sometimes be just one of us available.

Slides by D. Gries, L. Lee, S. Marschner, W. White

Recall: For Loops

```
# Print contents of seq
x = seq[0]
print x
x = seq[1]
print x
…
x = seq[len(seq)-1]
print x
```

The for-loop:

```
for x in seq:
    print x
```

- Key Concepts
  - **loop sequence:** seq
  - **loop variable**: x
  - **body**: print x
  - Also called **repetend**

Iteration: Doing things repeatedly

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

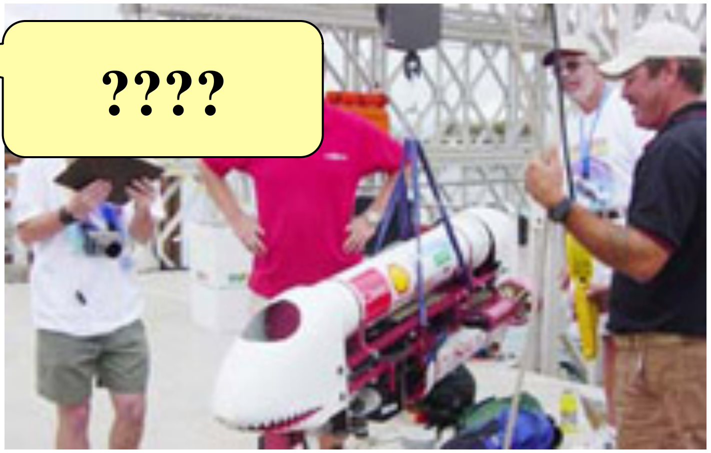
```
for x in sequence:
    process x
```

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

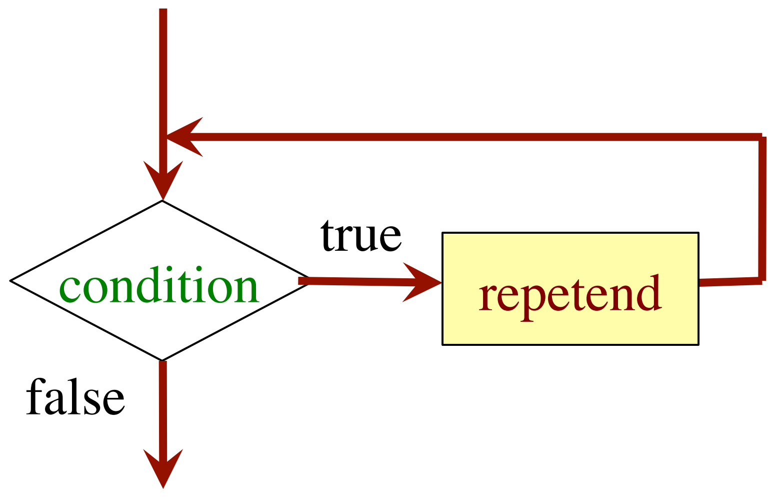
????



Beyond Sequences: The **while-loop**

```
while <condition>:
    statement 1
    …
    statement n
```

repetend or **body**



- Relationship to **for** loop
  - Broader notion of "still stuff to do"
  - Must ensure condition eventually becomes false
  - *You* explicitly manage what changes per iteration

While-Loops and Flow

```
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
End loop
Start loop 1
End loop
Start loop 2
End loop
After while
```

while Versus for

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

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

Must remember to increment

```
# 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
```

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 ???

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

while Versus for

Have to know in advance where to stop

```
# ble of squares to N
s q = []
n = floor(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
```

while Versus for

Fibonacci numbers:

$$F_0 = 1$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Sometimes you don't use the loop variable at all

```
# Table of n Fibonacci nums
fib = [1, 1]
for k in range(2,n):
    fib.append(fib[-1] + fib[-2])
```

Don't need to have a loop variable if you don't need one

```
# Fibonacci table up to N
fib = [1, 1]
while fib[-1] + fib[-2] < N:
    fib.append(fib[-1] + fib[-2])
```

Cases to use **while**

Sometimes you want to modify the sequence

```
# Remove all 3's from list t
i = 0
while i < len(t):
    # no 3's in t[0..i–1]
    if t[i] == 3:
        del t[i]
    else:
        i += 1
```

Stopping point keeps changing

Maybe this one is easier with no numerical counter

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

Cases to use **while**

Sometimes your termination condition has nothing to do with counters

```
def sqrt(c):
    x = c/2
    while abs(x*x – c) > 1e-6:
        x = x / 2 + c / (2*x)
    return x
```

Patterns for Processing Integers

range a..b-1

```
i = a
while i < b:
    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]
```

range c..d

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

```
# Store in double var. 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
```