Lecture 22

While Loops
Announcements for This Lecture

Assignments

• A6 due on **Wednesday**
  ▪ First two should be done
  ▪ Start Algorithm by weekend
  ▪ **Next Week**: Partition/Update
• A7 will be last assignment
  ▪ Will talk about next week
  ▪ Posted on Wednesday
• There is lab next week
  ▪ **No lab** week of Turkey Day

Prelim 2

• **TONIGHT**, 5:15 or 7:30
  ▪ **K – Z** at 5:15pm
  ▪ **A – J** at 7:30 pm
  ▪ See website for room
  ▪ Conflicts received e-mail
• Will have 4-5 questions
  ▪ Might drop short answer
  ▪ Similar to previous years
• Graded by the weekend

11/8/18
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
Important Concept in CS: 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

2. Perform \( n \) trials or get \( n \) samples.
   - A4: draw a triangle six times to make a hexagon
   - 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

---

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

```
for x in range(n):
    do next thing
```

????
Beyond Sequences: The *while*-loop

**while** `<condition>`:
- statement 1
- ...
- statement n

- Relationship to *for*-loop
  - Broader notion of “still stuff to do”
  - Must explicitly ensure condition becomes false
  - *You* explicitly manage what changes per iteration
print('Before while')

count = 0

i = 0

while i < 3:
    print('Start loop ' + str(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-l

for k in range(b,c)
    process k

Must remember to increment

# process range b..c

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

# process range b..c-l

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

# process range b..c

k = b
while k <= c:
    process k
    k = k+1
Range Notation

- \( m..n \) 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
Range Notation

• \( m..n \) 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 \( m \leq n+1 \)
  - So you can assume that even if we do not say it
  - If \( m = n+1 \), the range has 0 values
**while Versus for**

# incr seq elements

```python
for k in range(len(seq)):
    seq[k] = seq[k] + 1
```

# incr seq elements

```python
k = 0
while k < len(seq):
    seq[k] = seq[k] + 1
    k = k + 1
```

Makes a **range object**.

**while is more flexible, but requires more code to use**
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 = 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 = 1
while i <= n:
    v = v + 1.0 / i
    i = i + 1
# v= 1/1 + 1/2 + ...+ 1/n
while Versus for

# table of squares to N
seq = []
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

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

Fibonacci numbers:

\[
\begin{align*}
F_0 &= 1 \\
F_1 &= 1 \\
F_n &= F_{n-1} + F_{n-2}
\end{align*}
\]

# Table of n Fibonacci nums

```python
fib = [1, 1]
for k in range(2, n):
    fib.append(fib[-1] + fib[-2])
```

Sometimes you do not use the loop variable at all

Do not need to have a loop variable if you don’t need one

# Table of n Fibonacci nums

```python
fib = [1, 1]
while len(fib) < n:
    fib.append(fib[-1] + fib[-2])
```
Cases to Use **while**

Great for when you must **modify** the loop variable

```python
# 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 = i+1

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

11/8/18
While-Loops
Great for when you must **modify** the loop variable

```python
# 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 = i+1
```

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

Stopping point keeps changing.

The stopping condition is not a numerical counter this time. Simplifies code a lot.
Cases to Use while

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

```python
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
```

11/8/18 While-Loops 16
Welcome to CS 1110 Blackjack.
Rules: Face cards are 10 points. Aces are 11 points.
   All other cards are at face value.

Your hand:
2 of Spades
10 of Clubs

Dealer's hand:
5 of Clubs

Type h for new card, s to stop:

Play until player stops or busts
Welcome to CS 1110 Blackjack.
Rules: Face cards are 10 points. Aces are 11 points. All other cards are at face value.

Your hand: 2 of Spades 10 of Clubs

Dealer's hand: 5 of Clubs

Type h for new card, s to stop: Play until player stops or busts

How do we design this as a loop?
halted = False

while not game.playerBust() and not halted:
    # ri: input received from player
    ri = input('Type h for new card, s to stop: ')

    halted = (ri == 's')

    if (ri == 'h'):
        game.playerHand.append(game.deck.pop(0))
        print('You drew the ' + str(game.playerHand[-1]) + '\n')
Recall Lab 9

halted = False

while not game.playerBust() and not halted:
    # ri: input received from player
    ri = input('Type h for new card, s to stop: ')
    halted = (ri == 's')

if (ri == 'h'):
    game.playerHand.append(game.deck.pop(0)
    print('You drew the ' + str(game.playerHand[-1]) + '\n')
halted = False

while not game.playerBust() and not halted:
    # ri: input received from player
    ri = input('Type h for new card, s to stop: ')
    halted = (ri == 's')

    if (ri == 'h'):
        game.playerHand.append(game.deck.pop(0))
        print('You drew the ' + str(game.playerHand[-1]) + '\n')
Using **while-loops** Instead of **for-loops**

### Advantages
- Better for **modifying data**
  - More natural than range
  - Works better with deletion
- Better for **convergent tasks**
  - Loop until calculation done
  - Exact steps are unknown
- Easier to **stop early**
  - Just set loop var to False

### Disadvantages
- Performance is **slower**
  - Python optimizes **for-loops**
  - Cannot optimize **while**
- **Infinite loops** more likely
  - Easy to forget loop vars
  - Or get stop condition wrong
- **Debugging** is harder
  - Will see why in later lectures
Our Goal From Here: Sorting

Will see how to do this with while-loops