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

while Versus for

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

Must remember to increment

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

Patterns for Processing Integers

range a..b-1

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

# store in counts # of ’/’s in string s
count = 0
while i < len(s):
    if s[i] == ’/’:
        count = count + 1
        i = i + 1
    else:
        # Store in double var. by the sum
        # 1/1 + 1/2 + ... + 1/n
        v = v + 1.0 / i
        i = i + 1

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

While-Loops and Flow

print 'Before while'
count = 0
i = 0
while i < 3:
    print 'Start loop ' + str(i)
    count = count + 1
    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

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

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:
- \( F_0 = 1 \)
- \( F_1 = 1 \)
- \( F_n = F_{n-1} + F_{n-2} \)

```python
# Table of n Fibonacci nums
fib = [1, 1]
for k in range(2, 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 += 1
```

### Recall Lab 9

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:

```
How do we design a complex while-loop like this one?
```

### Some Important Terminology

- **assertion**: true-false statement placed in a program to **assert** that it is true at that point
  - Can either be a comment, or an **assert** command
- **invariant**: assertion supposed to "always" be true
  - If temporarily invalidated, must make it true again
  - **Example**: class invariants and class methods
- **loop invariant**: assertion supposed to be true before and after each iteration of the loop
- **iteration of a loop**: one execution of its body

### Preconditions & Postconditions

- **Precondition**: assertion placed before a segment
- **Postcondition**: assertion placed after a segment

```
# x = sum of 1..n-1  
x = x + n  
n = n + 1  
# x = sum of 1..n-1
```

```
1 2 3 4 5 6 7 8
x contains the sum of these (6)
```

```
1 2 3 4 5 6 7 8
x contains the sum of these (10)
```

**Relationship Between Two**
- If **precondition** is true, then **postcondition** will be true