Beyond Sequences: The **while-loop**

```plaintext
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

**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**

```plaintext
range a..b-l
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
# 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 = i + 1
# v= 1/1   + 1/2 + …+ 1/n
```

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

### While-Loops and Flow

**print 'Before while'**

```plaintext
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
End loop
End loop
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:

\[
F_0 = 1 \\
F_1 = 1 \\
F_n = F_{n-1} + F_{n-2}
\]

- # Table of n Fibonacci nums
  - 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

### Cases to Use while

Great for when you must modify the loop variable

- # Remove all 3's from list t
  - i = 0
  - while i < len(t):
    - if t[i] == 3:
      - del t[i]
    - else:
      - i += 1
  - # Remove all 3's from list t
  - while 3 in t:
    - t.remove(3)

The stopping condition is not a numerical counter this time.
Simplifies code a lot.

### 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?
Play until player stops or busts

### Preconditions & Postconditions

- **precondition**
  - # x = sum of 1..n-1
  - x = x * n
  - n = n + 1
  - # x = sum of 1..n-1

- **postcondition**
  - x contains the sum of these

**Relationship Between Two**

If precondition is true, then postcondition will be true