Recall: For Loops

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

- **Remember**:
  - Cannot program …
  - Reason for recursion

The for-loop:

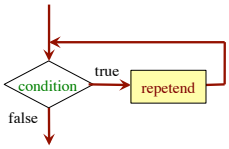
```
for x in seq:
    print x
```

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

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 explicitly ensure condition becomes false

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
```

Must remember to increment

- Makes list c+1-b elements
- List uses up memory
- Impractical for large ranges

- Just needs an int
- Much less memory usage
- Best for large ranges

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

```
# incr seq elements
for k in range(len(seq)):
    seq[k] = seq[k]+1
```

Makes a **second** list.

```
# incr seq elements
k = 0
while k < len(seq):
    seq[k] = seq[k]+1
    k = k+1
```

while is more flexible, but is **much tricker** 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 = 0
while i <= n:
    v = v + 1.0 / i
    i= i +1
# v= 1/1   + 1/2 + ...+ 1/n
```