Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers
    (either floats or ints)"
    result = 0
    result = result + thelist[0]
    result = result + thelist[1]
    ...
    return result
```

There is a problem here

Working with Sequences

- Sequences are potentially unbounded
  - Number of elements inside them is not fixed
  - Functions must handle sequences of different lengths
    - Example: sum([1,2,3]) vs. sum([4,5,6,7,8,9,10])
- Cannot process with fixed number of lines
  - Each line of code can handle at most one element
  - What if # of elements > # of lines of code?
- We need a new control structure

For Loops: Processing Sequences

```python
# 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

To execute the for-loop:
1. Check if there is a “next” element of loop sequence
2. If not, terminate execution
3. Otherwise, put the element in the loop variable
4. Execute all of the body
5. Repeat as long as 1 is true

Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers
    (either floats or ints)"
    result = 0
    for x in thelist:
        result = result + x
    return result
```

Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers
    (either floats or ints)"
    result = 0
    for x in thelist:
        result = result + x
    return result
```

For Loops and Conditionals

```python
def num_ints(thelist):
    """Returns: the number of ints in thelist
    Precondition: thelist is a list of any mix of types"
    result = 0
    for x in thelist:
        if type(x) == int:
            result = result + 1
    return result
```

```python
def num_ints(thelist):
    """Returns: the number of ints in thelist
    Precondition: thelist is a list of any mix of types"
    result = 0
    for x in thelist:
        if type(x) == int:
            result = result + 1
    return result
```
Modifying the Contents of a List

```python
def add_one(thelist):
    """(Procedure) Adds 1 to every element in the list
    Precondition: thelist is a list of all numbers
    (either floats or ints)""
    for x in thelist:
        x = x+1
    # procedure; no return
```

**Does Not Work!**

On The Other Hand

```python
def copy_add_one(thelist):
    """(Returns: copy with 1 added to every element
    Precondition: thelist is a list of all numbers
    (either floats or ints)""
    mycopy = []
    for x in thelist:
        x = x+1
        mycopy.append(x)  # add to end of accumulator
    return mycopy
```

Accumulator keeps result from being lost

For Loops and Frames

```python
def add_one(thelist):
    """(Procedure) Adds 1 to every element
    Pre: thelist is all numb.""
    for x in thelist:
        x = x+1

add_one(seq):
```

```python
def add_one(seq):
    """Add 1 to each elt"
    for x in seq:
        x = x+1
```

```python
0 1 2
```

```python
id4
```

Increments x in frame
Does not affect folder

For Loops: Processing Ranges of Integers

```python
The for-loop:
for x in range(2,201):
    total = total + x*x
```

```python
The range function:
\begin{itemize}
    \item range(x): List of ints 0 to x-1
    \item range(a,b): List of ints a to b-1
\end{itemize}
```

```
• The range function:
  \begin{itemize}
  \item range(x):
        List of ints 0 to x-1
  \item range(a,b):
        List of ints a to b-1
  \end{itemize}
```

```
1
2
3
```

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.
   - \( A_4 \): 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

```python
size = len(thelist)
for k in range(size):
    thelist[k] = thelist[k]+1
# procedure; no return
```

```
size = len(thelist)
for k in range(size):
    thelist[k] = thelist[k]+1
# procedure; no return
```

WORKS!