Example: Summing the Elements of a List

```
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

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

- **Remember**:
  - We cannot program …

The for-loop:

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

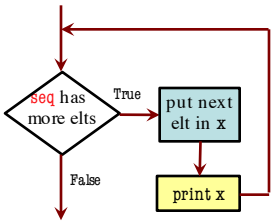
- Key Concepts
  - **loop sequence:** seq
  - **loop variable**: x
  - **body**: print x
  - Also called **repetend**

For Loops: Processing Sequences

The for-loop:

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

- **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

```
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
```

Accumulator variable

- **loop sequence:** thelist
- **loop variable**: x
- **body**: result=result+x

For Loops and Conditionals

```
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 the list:
        if type(x) == int:
            result = result+1
    return result
```

Body

Modifying the Contents of a List

```
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
```

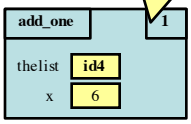
DOES NOT WORK!

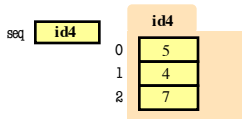
For Loops and Call Frames

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

add_one(seq):

Loop **back** to line 1





Increments x in **frame**
Does not affect folder

On The Other Hand

```
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 = []  # accumulator
    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: Processing Ranges of Integers

```
total = 0;
# add the squares of ints
# in range 2..200 to total
total = total + 2*2
total = total + 3*3
...
total = total + 200*200
```

- For each x in the range 2..200, add x*x to total

The for-loop:

```
for x in range(2,201):
    total = total + x*x
```

- **The range function:**
  - range(x): List of ints 0 to x-1
  - range(a,b): List of ints a to b-1

Modifying the Contents of a List

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

WORKS!

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